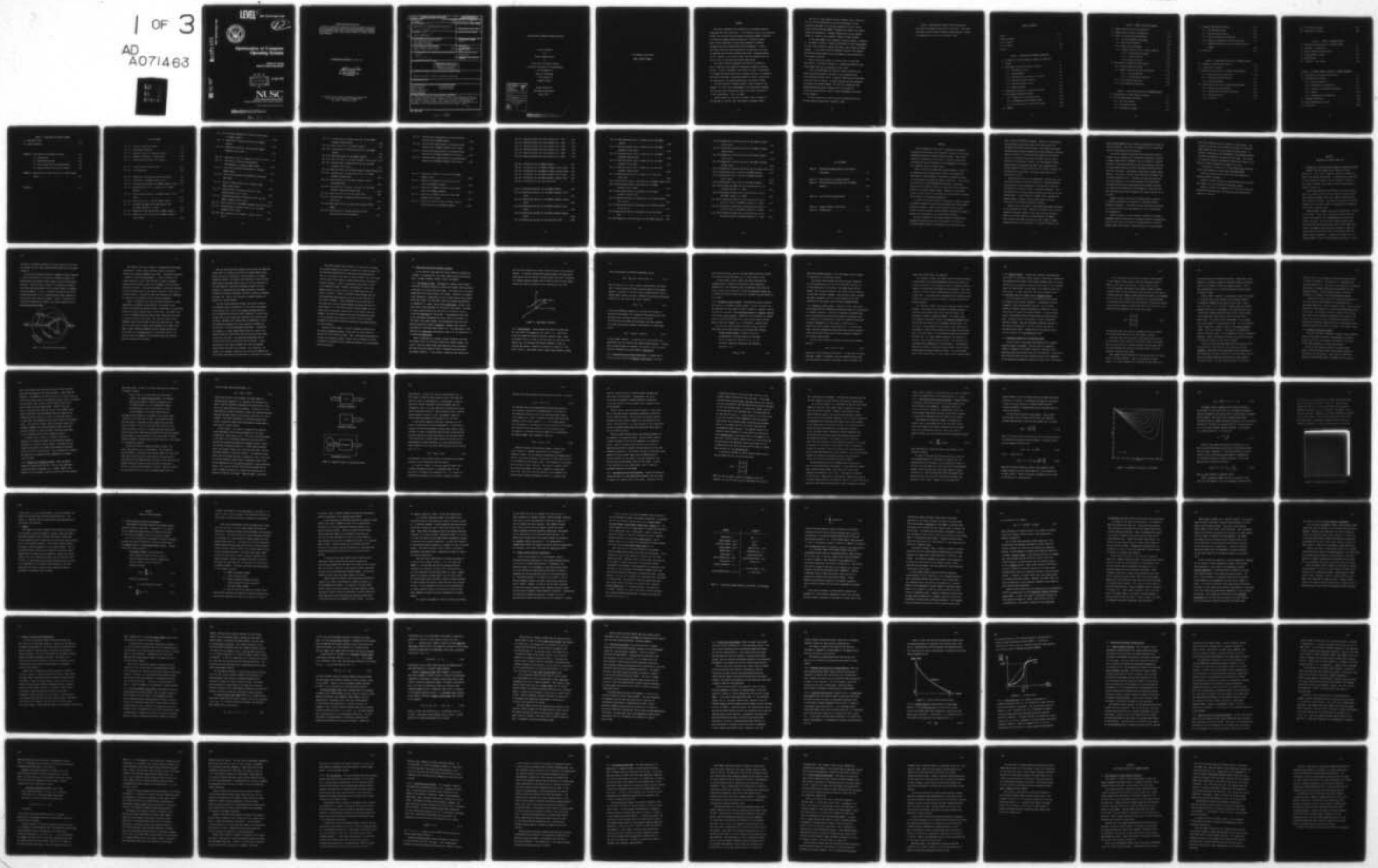
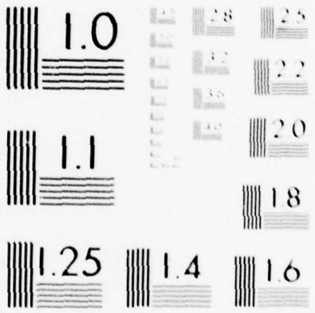
AD-A071 463 NAVAL UNDERWATER SYSTEMS CENTER NEW LONDON CT NEW LO--ETC F/G 9/2
OPTIMIZATION OF COMPUTER OPERATING SYSTEMS.(U)
APR 79 C R ARNOLD
UNCLASSIFIED NUSC-TR-6045 NL

1 OF 3
AD
A071463





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NUSC Technical Report 6045

NUSC Technical Report 6045

Optimization of Computer Operating Systems

Charles R. Arnold
Special Projects Department

15 April 1979

NUSC
Naval Underwater Systems Center
Newport, Rhode Island • New London, Connecticut

Approved for public release; distribution unlimited.

ORIGINAL CONTAINS COLOR PLATES: ALL DDC REPRODUCTIONS WILL BE IN BLACK AND WHITE.

ADMINISTRATIVE INFORMATION

This report is based on the unaltered Ph.D. thesis of Charles R. Arnold submitted to Harvard University. Much of the research reported in this document was made possible through support in previous years by the Naval Sea Systems Command (SEA 034) under Program Element 62721N, Mr. Phillip J. Andrews, Program Manager.

REVIEWED AND APPROVED: 15 April 1979

W. A. VonWinkle

W. A. VonWinkle
Associate Technical Director
for Technology

The author of this report is located at the New London
Laboratory, Naval Underwater Systems Center,
New London, Connecticut 06320.

14 NUSC-REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-6045	2. GOVT ACCESSION NO.
	3. RECIPIENT'S CATALOG NUMBER 9 Doctoral thesis
4. TITLE (and Subtitle) 6 OPTIMIZATION OF COMPUTER OPERATING SYSTEMS .	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Charles R.aymond Arnold	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Underwater Systems Center New London Laboratory New London, CT 06320	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62721N
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Underwater Systems Center Newport, Rhode Island 02840	11 12. REPORT DATE 15 April 1979
	13. NUMBER OF PAGES 250
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) 12 285p.	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Operating System
Control Theory
Resource Allocation
Program Behavior
Memory Management
Covariance Model
Optimal Predictor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The first portion of this thesis is devoted to an adequate formulation within modern control-theoretic concepts of the resource management function of computer operating systems. Then a major portion of the remainder of the thesis is devoted to exploiting estimating and control techniques for the management of the single most important resource, namely, memory.

405918

1B

OPTIMIZATION OF COMPUTER OPERATING SYSTEMS

A thesis presented

by

Charles Raymond Arnold

to

The Division of Applied Sciences

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Computer Science

Harvard University

Cambridge, Massachusetts

March, 1979

To the Memory of My Mother

AGNES (GARVIN) ARNOLD

PREFACE

This thesis represents the culmination of my graduate education spread over more than twenty years. Since computer science, sans operating systems, has only more recently become a recognized academic discipline, it reflects my somewhat broader perspective. Certainly, my many years of work experience at using various computers in numerous estimation and control applications should be apparent. In fact, this thesis might best be characterized as an about-face on my part. It represents the view that estimation and control techniques should aid the computer in its function rather than the computer being just an aid or tool in estimation and control applications.

This thesis cannot be regarded in any sense as a definitive work in applying control-theoretic techniques to computer operating systems. It truly represents a very small first step. Nevertheless, it is hoped that much of the flavor, structure, and power of estimation and control techniques in operating systems will remain revealed here for some time and will motivate others who I am sure will follow.

I am very grateful to numerous people — most of whom will go unnamed — for their help, encouragement, and intellectual influence over the many years and especially during the research reported on in this dissertation. To all, my thanks.

Special thanks are first due to Professor John J. Donovan of MIT who made me "see the light" and convert to computer science.

Most of all I must thank my advisor, Professor Ugo O. Gagliardi, for his initial receptiveness to my varied background, for his continuing confidence in me, and most importantly for his insight, direction and discerning comments throughout the conduct of my thesis research and preparation. Professor Gagliardi has won my complete respect as a master at the conduct of computer science research.

To the other members of my committee, Professors Thomas E. Cheatham, Jr. and Thomas C. Bartee, I express my gratitude for their encouragement, for their time invested in reading this thesis, and for their pertinent comments. I also credit Professor Cheatham for much of the congenial and stimulating environment within the computer science group which I have enjoyed at Harvard.

Special thanks go to Stuart W. Galley of MIT's Project MAC whose effort — described in Chapter 3 — produced the empirical data (trace tapes) which was so necessary for this study.

I must also acknowledge the financial support provided on two accounts by the Department of the Navy. As an employee of the Naval Underwater Systems Center, I received (long-term) training funds to cover my initial studies. The thesis research has been supported by the Naval Sea Systems Command. Mr. Phillip J. Andrews of Code NAVSEA-034 deserves special recognitiion for his efforts of justifying and maintaining financial support throughout the conduct of this research.

The final manuscript, as well as several preliminary versions, has been expertly typed by Mrs. Kathleen A. Buda.

Finally, I thank my wife, Kathy, for her patience and understanding throughout these often trying times. To my young son, Marty, who has neither patience nor understanding, I promise to somehow make up for the lost piggy-back rides.

TABLE OF CONTENTS

Chapter 1 BACKGROUND AND PROBLEM FORMULATION

Chapter 2 SURVEY OF RELATED RESEARCH

Chapter 3 FIRST ORDER STATISTICS OF PROGRAM BEHAVIOR

Chapter 4 SECOND ORDER STATISTICS OF PROGRAM BEHAVIOR

Chapter 5 A MODEL OF PROGRAM BEHAVIOR

Chapter 6 A CONTROL-THEORETIC APPROACH TO MEMORY MANAGEMENT

Chapter 7 DIRECTIONS FOR FUTURE RESEARCH

LIST OF FIGURES

LIST OF TABLES

SYNOPSIS

The first portion of this thesis is devoted to an adequate formulation within modern control-theoretic concepts of the resource management function of computer operating systems. Then a major portion of the remainder of the thesis is devoted to exploiting estimating and control techniques for the management of the single most important resource, namely, memory.

After a review of the operating system literature in 1974, it was determined that although several authors had suggested the application of control-theoretic techniques to operating systems, no one had given the problem an adequate formulation. This omission was rectified in a paper [ArG74] which provided a state-space formulation for operating systems and identified the job scheduling problem as an optimal control problem. Chapter 1 of this thesis contains a brief review of state-space and control-theoretic concepts and the details of the problem formulation.

In light of the control-theoretic formulation, Chapter 2 provides a reasonably complete survey and review through 1976 of the related published literature. Specific areas addressed are the general resource allocation problem, control-theoretic formulations, models of program and system behavior, and optimization techniques.

An early consequence of the control-theoretic formulation and subsequent literature survey was the recognition of the lack of validated dynamical models of the behavior of an operating system

from a resource utilization viewpoint. Moreover, past measurements upon operating systems had not been of the type required for validating the desired control-theoretic input-output system models. The application of modern estimation and control-theoretic techniques requires statistics of the second-order probability distribution such as the correlation, covariance, and power spectral density functions of the processes involved. Past measurements had been guided primarily by queueing type models and consequently were directed toward first-order statistics such as means, variance, and distributions of job arrivals, job delays, page reentry rate, working-set size, etc.

Realizing that a full set of measurements (for both first- and second-order statistics) for all resources of a complete operating system would be a major undertaking, it was decided to limit the investigation to a single resource — the memory component. Even the acquisition of a limited amount of memory usage data was no small accomplishment. It was realized in computer tapes containing the complete address reference string of seven complete (traced) execution of five different operational programs.

Chapter 3 gives the details of the program measurement experiment and the acquired trace tapes. Moreover, to provide a basis for later comparison, extensive analysis of the trace tapes was performed. Numerous first-order statistics are presented for the popular least recently used (LRU) and working-set (WS) schemes of memory management applied to the trace data. Our results contradict the

Denning-Graham [DeG75] view of comparative performance of these two popular algorithms. The results also demonstrate the non-Gaussian nature of the working-set size statistic.

The primary purpose for acquiring the trace data described in Chapter 3 was to provide a basis from which second-order statistics of memory usage could be estimated. As such, Chapter 4 is devoted primarily to the proper definition of the auto-covariance statistics of program memory behavior and their subsequent estimation from the empirical trace tapes. Two different perspectives of memory usage emerges — one from an individual page usage basis and another from a locality (ensemble) basis. Correlation and covariance functions from both perspectives are defined and shown to be reconcilable. The proper definition of the autocovariance of program localities also required the introduction of the concept of a "mean locality" which has been found to be an important first order statistic in its own right.

Chapter 5 presents the results of applying Prony's method to fit analytic expressions to the empirical covariance function estimates of Chapter 4. The prime conclusion being that the majority of covariance functions are approximated by a sum of two decaying real exponentials.

Chapter 6 provides a control-theoretic solution of the memory management function in the form of a predictor-controller decomposition. Using the empirically derived model of program behavior, classical optimal Wiener filter theory is applied to derive a single predictor to be used uniformily across all pages of a given program. The subsequent controller uses a threshold value upon the individual (page) predicted value to control specific paging actions. A simulated implementation (called OPTMEM) of the proposed predictor-controller was applied to the empirical trace tapes. The results are compared to Denning's working-set scheme and are shown to be as good as or better than Denning's scheme in all cases.

The material in Chapters 4-6 is new although some preliminary results were reported in earlier papers [Arn75],[ArG76].

Additional problems and direction for future research are addressed briefly in Chapter 7.

Appendix A presents derivations and additional results omitted from Chapter 1 on cost functions for operating systems.

Appendix B contains the complete set of first order statistics estimated from the empirical trace tapes which was described and summarized in Chapter 3.

CHAPTER 1

BACKGROUND AND PROBLEM FORMULATION

1.1 The Need for a Control-Theoretic Approach to Operating Systems

Computer operating systems can be viewed from several perspectives. From a user's viewpoint, they may be described in terms of the services they provide. A second and very useful view is that an operating system is first and foremost a manager of the system's resources such as memory, central processing units, channels, devices, etc.

Current theory permits the formulation of efficient queueing models to represent the flow of work through the various servers constituting the computer systim (viz. [Buz71], [BuS74] and the texts [CoD73] and [Kle76]). Such models are capable of predicting response time, utilization levels for processors, and through-put sensitivity to processor speed.

Queueing models, however, have serious shortcomings in several critical areas. First, their "state-space" does not model the levels of utilization or the interdependencies for all of the system's resources. Even the complex queueing network models which treat the CPU and other devices have a "state-space" which reflects the number of jobs and their distribution amongst the various queues of the network cannot simultaneously model the memory resource requirements. Secondly, the "control laws" for queueing models — their so-called queueing disciplines — are not functions of the demand parameters for system resources of the jobs on the queue, but are rather characterizing properties of the queues themselves.

The view of an operating system as a manager of system resources is treated extensively in the recent text [MaD74] by Madnick and Donovan wherein they devote a major chapter to each of the system's resources. There the author's approach is deliberately pragmatic since there does not presently exist a unifying theory for computer resource management. In fact, the Madnick and Donovan overview of an operating system is still a network queueing model expressed as their process state transition diagram. Figure 1-1 presents such a transition diagram which, however, this author chooses to call a <u>job state transition diagram</u>.

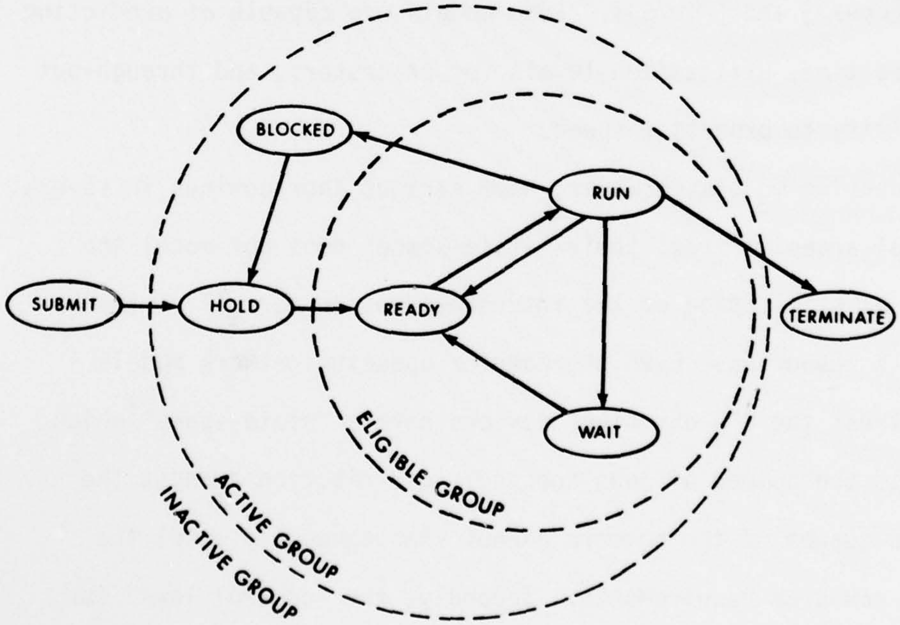


FIGURE 1-1 Job State Transition Diagram

The choice of "job" over "process" is prompted by the following consideration. Within current operating systems, the system's resources are typically managed at two levels. The overall supervisor — generally called the job scheduler — assigns eligibility for system resources to a job. Once eligible, a group of secondary schedulers and allocators control the assignment of the individual resources to the job. Since the job scheduler is the major controller of the utilization levels of the system's resources; it will be our principle concern. The choice of jobs over processes thus emphasizes our concern for the dominant level of resource management.

Figure 1-1 presents a job state transition diagram for a system comparable to the MULTICS System [Org72]. The progress of any specific job through the operating system is modeled by its transitions between the various queueing states of the figure. The SUBMIT state is for all submitted batch jobs and for those interactive users who are trying to log onto the system. The remaining states, except TERMINATE, model various phases of the active jobs and users. Moreover, for the system to support multiprogramming on a paged virtual memory without thrashing [Den68c], one must further restrict the active users to an eligible group as in the MULTICS System. At any given time, only members of the eligible group are given the physical resources of the system.

With the state transition diagram as an overview, the operating system itself is viewed as a collection of program modules which control the various transitions a job will make in its progress through the system. For example, the transitions of jobs from the SUBMIT state to the HOLD state (Figure 1-1) will be controlled by the Load Scheduler; transitions from HOLD to READY by the Job Scheduler in conjunction with the Memory Allocator; from READY to RUN by the Processor Scheduler; from WAIT to READY by the Memory Allocator; etc. Most of the allocator or scheduler modules are managers of a single resource.

For many current operating systems, the various allocators and schedulers make no assumptions as to the actual requirements of any specific job. With little a priori knowledge (possibly user assigned priority and job class), scheduling is usually done on some cyclic basis (round-robin) possibly within some priority structure. Moreover, most systems make no assumptions or use of the interactions between the various resources or between the various users in the multiprogramming environment. The MULTICS job scheduler is a good example wherein jobs are raised to the eligible group on a cyclic basis from the top of a priority queue. A given job's position in the queue being determined from its initial priority and its recent past CPU utilization. A major shortcoming of this type of scheduling is that the scheduler cannot balance the total active demands. Such schedulers, for example, will sometimes introduce a job with high CPU demand into the eligible group which may already have a high degree of CPU demand.

Motivated by modern control theory, it is felt that a solution to the above problem is to provide a centralized resource manager for the operating system which will have three prime characteristics. First, at all times it will have a current estimate of the utilization level for each of the system's resources being used by the group of eligible processes — that is, an estimate of the current "state" of the operating system. Secondly, the system will collect and update statistics on each job as to its "demand" made upon the system's resources. Thirdly, the global resource manager will use some system performance measure in its task to meet the goals of the system's designers. From a knowledge of the current state of the system and a set of estimated demands associated with jobs in a holding queue, it is felt that an overall resource manager can do a better job at maintaining a balanced use of the system's resources. Moreover, the control of response time within the operating system is not disconnected from resource utilization levels but becomes an adjunct to the overall problem of specifying a performance measure (or cost function).

The goal of this chapter is to give an adequate formulation to the resource management problem. State-space and control-theoretic concepts are first briefly reviewed. Then a state-space representation is given to an operating system. Finally, the job scheduling problem is defined as an optimal control problem and analogies are drawn between the two disciplines.

1.2 State-Space and Control-Theoretic Concepts

In this section, state-space and control theoretic concepts are reviewed. For more details, the reader should consult the following texts: [AtF66], [BrH69], [Ast70], [Kus71], and [McG74].

1.2.1 The Concept of State. The state of a system is that minimum sufficient information required to predict the system's future evolution on the basis of current and future inputs. Past inputs are, in essence, summarized by the current state of the system. Typically, the required state information is expressed as a set of values taken on by variables within the system. The individual values are ordered and identified as components of a vector, the so-called state-vector. Values of the state vector being synonymous for the state of the system. The range of values possibly taken on by the state as a function of time is called the state-space for the system. A representative point $\underset{\sim}{x}(t)$ in the state-space corresponds to a given state of the system. As the state of the system changes, the representative point $\underset{\sim}{x}(t)$ will describe some curve called a trajectory or motion (terms inspired by dynamics) in the state-space (Figure 1-2). The state-space is also referred to as phase-space and especially when it is two dimensional, it is called the phase-plane.

Which and how many of a system's internal variables constitute the system's state is a difficult question for complex systems. The result is that one usually makes some subjective choice dictated by the types and by the degree of detail of questions one poses about the system's behavior. If the system is subject to some conservation law, the state variables may reflect the partitioning of the conserved resource. In physical systems which conserve energy, the state-vector often express the distribution of energy amongst the system's components. For computer operating systems, the utilization levels for each system resource are factors in meeting a total computational work-load.

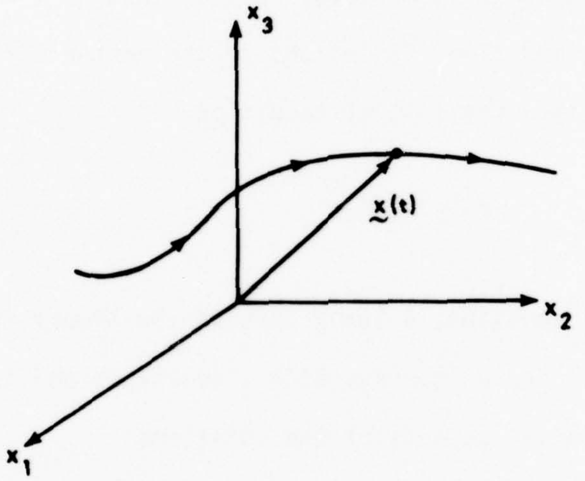


FIGURE 1-2 State-Space Trajectory

1.2.2 System Dynamics. Having defined the state of a system, one must next specify the dynamics of the system, i.e., a description of how the system will evolve in time for a specific input. Given the system's state $\underset{\sim}{x}_0$ at time t_0 and some input $\underset{\sim}{u}(t)$ over the closed interval $[t_0, t_1]$, one needs the system's dynamics in order to calculate the system's trajectory or evolution of states to a final state of time t_1. Much modern control theory treats dynamical systems which are governed by differential equations such as

$$\dot{\underset{\sim}{x}}(t) = \frac{d}{dt}\,\underset{\sim}{x}(t) = \underset{\sim}{f}(\underset{\sim}{x}(t),\ \underset{\sim}{u}(t),\ t). \qquad (1.1)$$

Here the state $\underset{\sim}{x}(t)$ is a real n-vector valued function; the control $\underset{\sim}{u}(t)$ is a real m-vector for each t; and the essence of the system's dynamics is contained in the real n-vector valued function $\underset{\sim}{f}$ of the current state, control, and time. Solutions to the vector differential equation (1.1) must satisfy the initial condition

$$\underset{\sim}{x}(t_0) = \underset{\sim}{x}_0. \qquad (1.2)$$

As for any differential equation, a large part of the theory is devoted to the existence and uniqueness of the solutions and to the qualitative and quantitative aspects of the solutions.
The difficulties associated with the general non-linear equation (1.1) often forces one to consider linearizations or linear models such as

$$\dot{\underset{\sim}{x}}(t) = A(t)\underset{\sim}{x}(t) + B(t)\underset{\sim}{u}(t) \qquad (1.3)$$

for the system's dynamics. In equation (1.3), A(t) and B(t) are, respectively, an n by n and an m by m matrix-valued function. If these matrix-valued functions are constant, the system is said to be time-invariant; otherwise we say the system is time-varying.

1.2.3 Controllability and Control Constraints. A system such as (1.1) or (1.3) is said to be completely controllable if for each given initial value $\underset{\sim}{x}_0$, one can find some control which will transfer the system to any desired state $\underset{\sim}{x}_d$ in a finite length of time.

In many systems, control signals are usually obtained from equipment which is subject to saturation or other constraints. Mathematically, such constraints must be modeled by restricting the control vector $\underset{\sim}{u}(t)$ to some set U, called the admissible control set. It follows that control restrictions will influence the controllability of a system.

1.2.4 The Optimal Control Problem. Each admissible control function $\underset{\sim}{u}(t)$ results in a different system response. In order to select from the admissible set that control u* which is "best" or "optimal" requires the definition and use of some performance measure or objective function (also called a cost function). Objective or cost functions for complex systems are often quite subjective in that one must usually counterbalance conflicting subgoals (e.g., utilization v. response time).

For any given system objective functional $J[\underset{\sim}{u}|\underset{\sim}{x}_0]$, possibly conditioned by the initial state $\underset{\sim}{x}_0$, one can pose the following:

> OPTIMAL CONTROL PROBLEM - Subject to the given system dynamics, find that control $\underset{\sim}{u}^*$ from among the set of admissible controls U (i.e., $\underset{\sim}{u}^* \in U$) such that $\underset{\sim}{u}^*$ optimizes (maximizes) the objective function J, i.e.,

$$J[\underset{\sim}{u}^*|\underset{\sim}{x}_0] = \text{Max.} \tag{1.4}$$

When the performance measure is cost, the optimal control problem is formulated as a minimization problem.

The optimal control problem has three principal ingredients - (1) the system dynamics, (2) the set of admissible controls, and (3) the objective or cost function. The reader should realize, however, that the above statement of the optimal control problem is in some sense a minimal statement. Most practical control problems have other ingredients such as an output mapping and/or other constraints on the system's state and output [AtF66],[BrH69].

1.2.5 Stochastic Control Theory. Stochastic control theory deals with systems which are subjected to disturbances which are characterized as stochastic processes. The noise or stochastic disturbances may be (1) purely external and effect the system as the nominal input or as additions to a deterministic input, (2) internal arising from stochastic parameters or as self noise, or finally (3) as a combination of both internal and external noise. In spite of the actual situation, mathematical models which have only the (properly conditioned) additive input noise are quite adequate in modeling all the above cases [Ast70].

As such, one can model the stochastic system by the following equation

$$\dot{\underset{\sim}{x}}(t) = \underset{\sim}{f}(\underset{\sim}{x}, t) + \underset{\sim}{v}(t) \tag{1.5}$$

where $\underset{\sim}{v}(t)$ is the disturbing noise process. For the above continuous time model, however, as opposed to the corresponding discrete time case, the process $\underset{\sim}{v}(t)$ must be restricted in order to obtain valid models (see [Ast70] chap. 3 for details).

For stochastic systems, one cannot require the future evolution of the system to be uniquely determined by the current state $\underset{\sim}{x}(t)$. A natural extension of the concept of state of stochastic systems is, however, to require that the probability distributions of the state variable $\underset{\sim}{x}$ at future times be determined by the actual current value of the state. If one now recalls the definition of a Markov process [B-R60], one requires that such stochastic system be described as Markov processes.

For stochastic systems, one may also formulate optimal control problems. The solution of such problems for either discrete or continuous time systems relies heavily upon the concepts and techniques of dynamic programming [How60]. For linear systems with quadratic performance criteria, the Separation Theorem [Ast70] implies that the solution can be decomposed into two parts. The first part is the estimation of the systems's state, typically as the output of a Kalman filter [Kal60] which provides the conditional mean of the state subject to the observations made upon the system. The other part is a linear feedback problem for the control as a function of the estimated value of the state. The second part of the problem being independent of the disturbances. The separation theorem thus provides a connection between optimal filtering theory and the theory of optimal stochastic control. For every optimal linear filter problem there is a dual optimal control problem [KaB61].

1.2.6 Adaptive Control. In many real situations, the formulation and solution of an optimal control problem is impossible or impractical. The problem usually relates to the system's dynamics which, for either deterministic or stochastic systems, are often either (1) not completely known, (2) time-varying, (3) non-linear, or (4) both non-linear and time varying. In these situations, the control engineer resorts to techniques which have become known as adaptive control.

For our purposes, one can view the adaptive control problem as having but one principal ingredient - a performance criterion or measure. With the performance functional, one can employ gradient or feasible direction techniques to determine a control input which will improve the systems response. As such, one must accept a sub-optimal solution but these again can often be quite good. How good or how far from optimal is often an impossible question to answer. One great advantage of adaptive solutions is that they are implemented as "on-line" solutions and need not anticipate future inputs. Systems with on-line adaptive control are also referred to as self-optimizing systems.

1.3 State-Space Concepts for an Operating System

In this section, a state-space representation for a computer operating system is developed. With the view of an operating system as a manager of the system's resources, the state of the operating system should reflect the use of the resources. Specifically, the utilization levels for each resource are used as the components of the state-vector, ordered by decreasing relative cost of the resource.

It will be helpful at this point to consider a simple example. Let the computer system consist of but three resources, namely, a memory, a CPU, and an I/O processor. At any instant of time, let $x_1(t)$ represent the fraction of the number of pages of memory currently being utilized (referenced within the past Δt time interval) divided by the total number of pages that physical memory can accommodate. Let $x_2(t)$ and $x_3(t)$, respectively, represent the current percentage of utilization for the CPU and the I/O processor in the past Δt interval. The state of the system can then be described by the state-vector $\underset{\sim}{x}(t)$ formed from the components $x_i(t)$ in the usual fashion,

$$\underset{\sim}{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} . \qquad (1.6)$$

The resulting state $\underset{\sim}{x}(t)$ giving us a composite view at any instant of the utilization levels of all of the system's resources.

One might argue that the system's state-vector should reflect the collection of jobs currently in the system's queue of ineligible jobs but as we shall see in the next section, the view taken here is that they represent the collection of possible inputs to the job scheduler of the operating system.

Most computer system resources are discrete and their utilization levels - our state variables - vary in a discontinuous fashion. However, it is often mathematically convenient to treat them all as continuous and even differentiable variables. Additionally, when treated as a stochastic system the individual resource utilization levels will be viewed as continuous random variables. Similar type considerations occur in the diffusion approximation [New71] to queueing systems.

Briefly, what is required is a macroscopic view of the system. To illustrate such a view, consider only the memory component. If the memory component has the value $x_1(t)$ at time t, then a short time later, $t + \Delta t$, the level of memory usage has probably changed at most by one or two pages. One should not think of an "infinitesimal" memory change but rather think of memory changes as being negligibly small compared with the current level of memory usage. On a coarse scale of measurement, it has changed only a small amount in a short period of time. This local behavior is well recognized in computer science as the locality property [CoD73]. Such local character - the value of a variable being related to adjacent values - is the underlying origin of differential equations. Hence, the property of locality also justifies the use of differential models for the system's dynamics.

Finally, the one remaining property that one requires of the state-variables is that they possess reliable and efficient estimators. Motivated by this requirement and yet maintaining a dual perspective of either a discrete or continuous model, one can make the following operational definition for system resource utilization levels. For each resource, define its utilization level at time t as the average utilization during some interval of length T preceding t. The choice of T is governed by the same considerations as is the choice of the window size in Denning's definition of a program's working set (and working set size) [Den68b], [DeS72]. Locality properties again justify the existence of a suitable T and the corresponding estimator.

Estimation of resource utilization levels was addressed in a very basic paper by Wulf [Wul69] in his consideration of a performance monitor for a multi-programming system. There the author makes many of the arguments for a state-space formulation but fails to give such. Even without an explicit concept of the state of the system, he does introduce an estimator for the state of the system. Wulf was also close to formulating his problem as an optimization problem but admits the lack of a "meaningful explicit objective function" which used his estimate of the system's state.

1.4 The Resource Allocation Problem

Given our state-space formulation in terms of the utilization levels of the system's resources, the management of resources becomes equivalent to the problem of the control of the state of the system, hopefully, in some optimal fashion.

Within current operating systems, the primary mode of control of the utilization levels of the system's resources is exercised by the job scheduler. Recall that it controls the transitions from the HOLD state to the READY state in Figure 1-1 and as such controls the transitions of jobs from the ineligible group with no resources to the eligible groups — those receiving resources. The composite utilization levels of all eligible jobs constitutes the current state of the operating system. If one is to control the utilization level, one must certainly have control of the entry of jobs into the eligible group. Control of only job entry is one-sided; more effective control of the system's state could be realized if the job scheduler were also capable of descheduling selected jobs so as to give it a two-sided control. In either case, job scheduling is the primary control on utilization levels. This does not, however, imply (or prevent) some of the secondary schedulers in current operating systems such as the processor scheduler or memory allocator from having some effect on the utilization level of their respective resource.

In order to get a "better" or "optimal" allocation of resources in an operating system, the job scheduling problem is formulated as an optimal control problem. From above, one can visualize the state of the system as the combined utilizations levels of the set of eligible jobs. From a deterministic viewpoint, one requires as a minimum the three components of an optimal control problem.

1.4.1 Dynamics for an Operating System. Under the general heading of "system identification", there has been developed a large collection of techniques (e.g., [Lee64], [SaM71], and [Eyk74]) for fitting models to empirical observations (measurements) made upon systems. Briefly, the system identification problem can be stated as follows:

> Given a class C of system models and some physical system S, the identification problem is to determine that specific model in C which is equivalent to S. The identification is to be accomplished through the observations, often in the presence of noise, of the response of S to various probe functions.

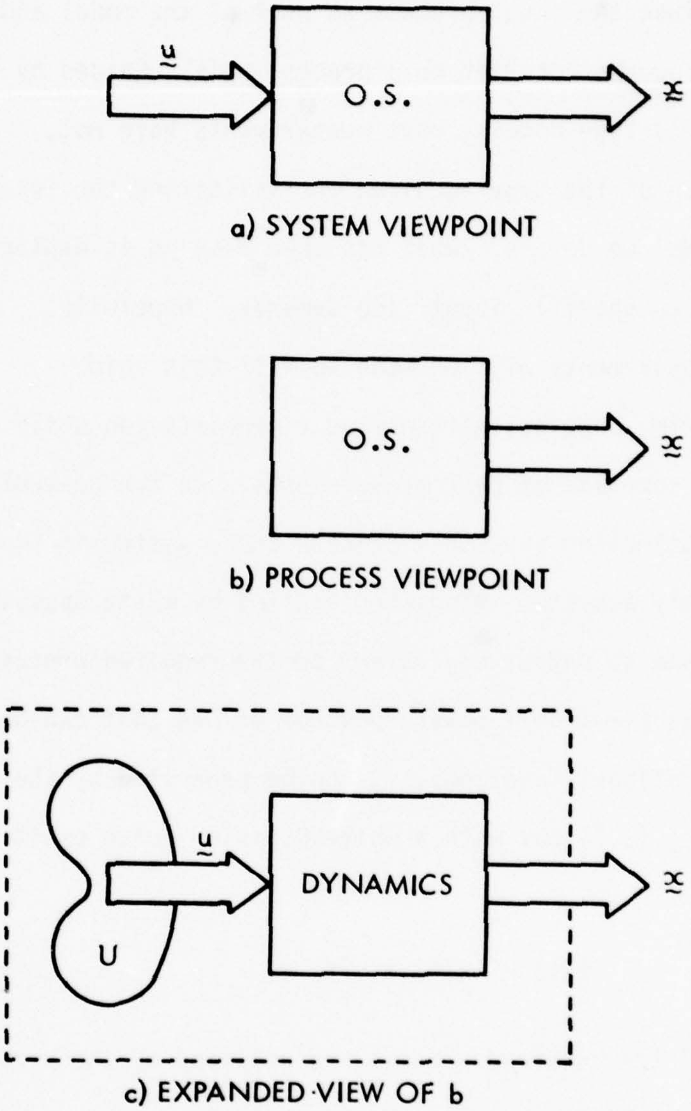
The identification problem is also known in the control theory literature as the "black box" problem. The connotation being that the system is equivalent to some unknown black box which cannot be opened. The only insight that one can obtain of the internal workings of the box is through experiments which probe the box with various inputs and which observe the resulting response. System examples are the estimation of the system Green's function (impulse response), frequency domain transfer function or coefficients of a differential equation.

Turning specifically to operating system, one must first select some general class of system models. The differential equation models in state-space form provide a good first choice over other model classes because of their ability to yield recursive algorithms which are computationally efficient in both time and space [BuJ68]. Differential models are also justified as observed in Section 1.3 by the locality property. Finally, simplicity dictates that one should first try to find an adequate model within the class of linear time-invariant model, e.g.,

$$\dot{\underset{\sim}{x}}(t) = A\underset{\sim}{x}(t) + B\underset{\sim}{u}(t), \qquad (1.7)$$

rather than from the class of general non-linear models as characterized by equation (1.1). One must still determine the model's specific parameters contained as the coefficients in the matrix A by some identification procedure. Initially, as a first approximation, one may also choose to ignore the interaction between resources and hence restrict the matricies A and B of (1.7) to be diagonal. For either the general or restricted diagonal case, one has a theoretical model which should govern the measurements made upon the computer system.

To date, however, this author is unaware of any operating system measurements which would be adequate for fitting and validating of a model such as (1.7). The lack of such measurements is the result of an alternative theoretical viewpoint which in the past has dominated the modeling of computer systems. Rather than viewing an operating system as a "system" which transforms inputs into outputs (Figure 1-3a), the prevailing models view the operating system as a stochastic "process". The operating system exists with its offered load and its evolution in time is described by a vector stochastic process which it generates (Figure 1-3b). From an expanded view of the operating system as a process generator (Figure 1-3c), one can see that the distinction between the two views is whether one chooses to consider the offered-job-mix as internal or external to the model. Queueing models - which have



a) SYSTEM VIEWPOINT

b) PROCESS VIEWPOINT

c) EXPANDED VIEW OF b

FIGURE 1-3 Dynamical Views of an Operating System

been dominant - by their very nature of specification (e.g., a M/G/1 queue) include the input process as part of the model and hence fall under our designation as a process model. Guided by theoretical process type models, past measurements have not, consequently, been of the type required for validating the input-output system model we desire. What has been missing is explicit system responses to specific input job demands. Hopefully, future system measurements will be made to fill this void.

In the interim, however, alternative viewpoints can still allow us to make some use of past measurements. To the control engineer, the distinction between a process and a system is small. A process is simply a system which when excited by white Gaussian noise will generate an output equivalent to the required process. If the process has a rational power spectrum or one that can be approximated by rational functions, it can be generated by the linear dynamics of (1.7) but with a white Gaussian noise excitation v(t), i.e., as

$$\dot{\underset{\sim}{x}}(t) = A\underset{\sim}{x}(t) + \underset{\sim}{v}(t). \tag{1.8}$$

The parameters of the required process are realized by the proper selection of the coefficients with the A matrix.

As a specific example, it has been suggested [CoR72] that an Ornstein-Uhlenbeck process is a reasonable model for the evolution of the memory state component $x_1(t)$. To a control or communications engineer, such a process is viewed as colored

Gaussian noise whose normalized auto-correlation function is given by

$$\phi_x(\tau) = \exp\{-\mu|\tau|\} \qquad (1.9)$$

The required process can be generated by passing white Gaussian noise through a low-pass filter [Syn69] which is realized as a one dimensional version of (1.8); the coefficient matrix being a single number, say α, which is directly related to the single free parameter of the process, namely $\mu = -\alpha$. As in Bryant [Bry75], past measurements of a working-set size can be used to first estimate the correlation time (the coefficient μ) of the process $x_1(t)$. Consequently, a differential dynamical model for the evolution of the system's memory state component is given by

$$\dot{x}_1(t) + \mu x_1(t) = v(t) \qquad (1.10)$$

Another area where the distinction between a process and a system viewpoint is somewhat reconcilable is in the area of stochastic control. By the separation theorem [Ast70] [BuJ68] [Kus71] [Gel74], the problem decomposes into two parts of which the first part is an estimation of the system's state. The estimation of the system's state can take a process viewpoint. The second or feedback control part can possibly neglect the system's dynamics or assume some simple form. While the separation theorem is only mathematically justified under quadratic performance criteria, it provides some heuristic justification for estimation/control decomposition under other critiera [Sch73]. Alternatively, one can go all the way and approach the whole problem as a prediction (filtering) problem with the assumption that subsequent control will be obvious.

Finally, one can view an operating system as a free-running most of the time and only occasionally subjected to controls at the job scheduling epochs. During these free-running intervals, one again may use a process viewpoint for the evolution of the system. One possibility is to view the motion of the system state-vector as a Brownian Motion with the state-vector being reset at the job scheduling epochs.

There are other possible process and system type models for the dynamics of an operating system. The next chapter presents a survey of models for the various components of an operating system. Some of the models given there may also be used to describe the evolution or dynamics for more than one of the state components (resources). The validity of models for operating system dynamics will have a major impact upon the determination of proper estimators for the system's state and subsequently upon our ability to provide proper control to the system. In spite of the importance of such system models, there is almost no published literature on the problem.

1.4.2 Job Demands and the Offered-Job-Mix. Viewing the operating system controller as a job scheduler/de-scheduler, the only inputs or controls are theactive jobs in the system. Therefore, the set of admissible controls is the set of jobs currently in the system's queues containing all the active jobs. If one takes the more restricted (one-sided) view that jobs are only scheduled (and not de-scheduled), then the admissible set would be the queue of only the ineligible jobs. In either case, however, one must still equate each of the individual jobs in the admissible set to some control vector $\underset{\sim}{u}_j$. A natural choice is to let the vector $\underset{\sim}{u}_j$ for each job have as many components as does the system's state vector; the individual values reflecting some estimated measure of the demand that the j-th job will make upon the corresponding system's resource. The composite vector of all component resource demands of a given job will be referred to as the <u>demand-vector</u> or even more briefly as the <u>demand</u> for the given job. Having assigned a demand vector $\underset{\sim}{u}_j$ to each active job, the equivalent of the admissible control set becomes the set of demand-vectors for the active jobs which for want of a better name, we shall call the <u>offered-job-mix</u>.

In the earlier example of a three resource system, one can view the demand for the j-th job as the vector,

$$\underset{\sim}{u}_j(t) = \begin{bmatrix} u_{1j}(t) \\ u_{2j}(t) \\ u_{3j}(t) \end{bmatrix} \qquad (1.11)$$

Recalling that the memory resource corresponds to the first component, one can also view $u_{1j}(t)$ as some measure of the j-th job's working-set-size [Den68b]. The other two components describe the job's demand for the CPU and the I/O processor, respectively.

As a practical matter, it is impossible to know the time-varying demand for each job in the system. What is required is a more practical measure of demand. Moreover, such measures of job demand should be capable of having efficient estimators [DeE71]. Several possibilities present themselves. First, one could remove the time-dependence by taking some total measure such as the integral of the demand over the duration of the job. In a similar vein, one could also take time averages over the duration of the job. Lastly, one might choose a demand vector whose components equal the maximum value realized by each corresponding resource over the duration of the job. In all cases, it can be shown that simple efficient estimators can be found which quantifies the various job demands as a by-product of the system's effort to continually estimate its own state. Jobs first entering the system may have to be given some mean demand-vector, but thereafter, as the job circulates between the eligible and ineligible groups, the system should improve (or refine) the estimate of each job's demand.

Having designated the set of demand-vectors for the active jobs as the offered-job-mix and equated it to the admissible control set of optimal control theory, this author will use the same symbology and represent it as U or U_t — the subscript to remind us that the set's composition is time-varying. Whether one views the individual demand vectors as time-varying, the mix is surely time-varying because of jobs entering and leaving the system. Analogous to stating that a specific control is in the admissible set, one can speak of a specific job demand being in the offered-job-mix, i.e., $\underset{\sim}{u}_j \varepsilon U_t$.

1.4.3. Cost Functions for an Operating System. Finding a measure of operating system effectiveness, as with any complex system, often involves value judgements. The measure itself can be cost, efficiency, through-put, response-time, profit, etc. Having selected some measure, one must also be able to express its quantitative dependence upon the system state variables.

One approach for operating systems is to express the system's cost as the weighted sum of the individual resource costs, each resource cost being a function of the utilization level of its corresponding resource. With $C_i(x_i)$ being a generic resource cost, one can write the total system cost function as

$$C(\underset{\sim}{x}) = \sum_{i=1}^{n} \alpha_i C_i(x_i) \qquad (1.12)$$

where the weights α_i reflect the actual cost (rentals) of the different resources.

For each of the normalized resource costs $C_i(x_i)$, one can use a weighted measure of efficiency vs. response time. By assuming that for each resource, its utilization will result in billing to the user which will recover its cost for the operation manager, one can see that each $C_i(x)$ should contain a term of the form (1-x). With no utilization (x=0), the operation may pay the full cost, but with full utilization (x=1=100%), the operation's cost is zero. However, as is well known from queueing theory, as the utilization levels go to 100%, the system delays go to infinity. To counteract this behavior, one should include in the cost functions a weighted second term which reflects the delays or overhead which results from high levels of resource utilization.

The CPU resource provides a good example. Using a M/G/1 queueing model with a job mean service time of one time unit, the mean queueing delay is given by (see [CoD73], eqs. (4.2.34)).

$$\bar{D}(x) = \frac{x(1 + C_v^2)}{2(1 - x)} \tag{1.13}$$

where x is the utilization factor and C_v is the so called coefficient of variation of the service time distribution. Using this result, one can represent the CPU cost as

$$C_i(x) = (1 - x) + \beta\bar{D}(x) \tag{1.14}$$

or better, (with i=2) as

$$C_2(x_2) = (1 - x_2) + \beta_{2k} \frac{x_2(1 + C_v^2)}{2(1 - x_2)} \tag{1.15}$$

where the fractional weight β_{2k} reflects some subjective value judgment between the cost of more hardware vs. having people wait for their results. Figure 1-4 presents a parametric set of curves of the CPU cost vs. utilization for

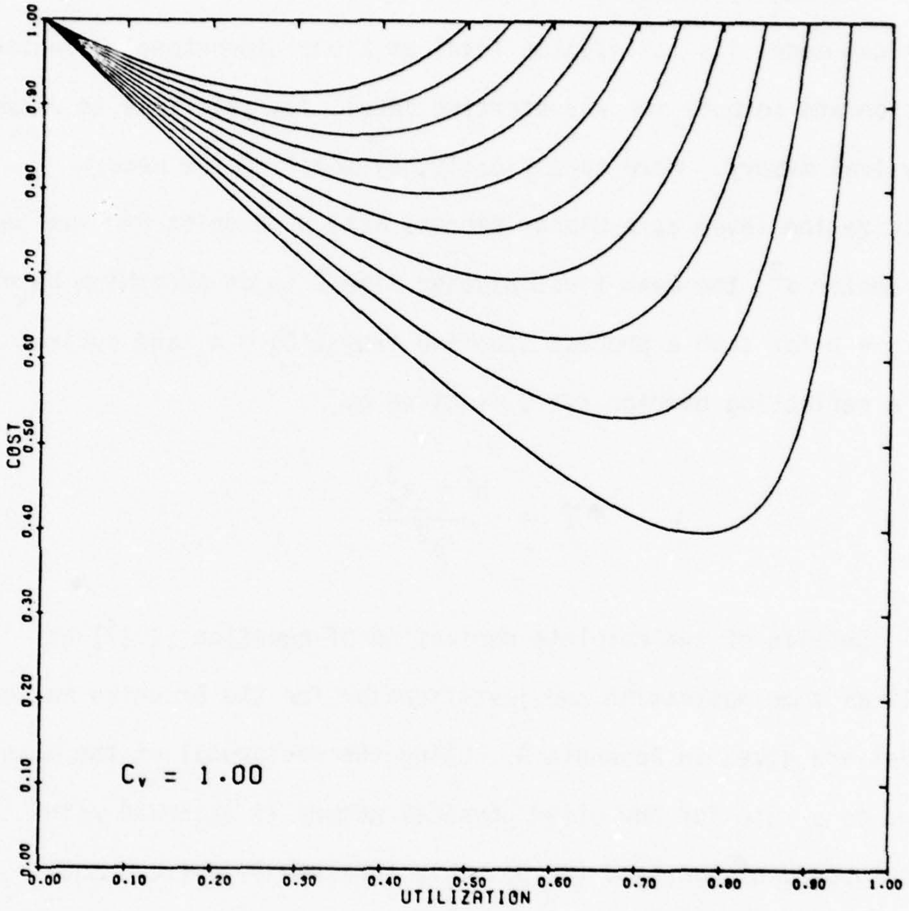


FIGURE 1-4 Parametric CPU Costs vs. Utilization

$$\beta_{2k} = 0.05k, \; k = 1, 2, \ldots, 10. \qquad (1.16)$$

The memory resource component can be treated differently. One can model its utilization level as a one dimensional Brownian motion and compute for any starting level, its mean time to exceeding physical memory. More specifically, by modeling the memory utilization level as a Wiener process $X(t)$ with drift $\mu=0$ and variance parameter σ^2, the mean first passage time $\bar{T}$ to an absorbing barrier at $x = b$ for such a process starting from $X(0) = x_o$ and subject to a reflecting barrier $x = 0$ is given by

$$\bar{T} = \frac{b^2 - x_o^2}{\sigma^2} \qquad (1.17)$$

Details of the complete derivation of equation (1.17) as well as some motivation and justification for the Brownian motion model are given in Appendix A. Using the reciprocal of the mean time as a rate for how often physical memory is exceeded with its subsequent overhead (or possible thrashing), one can model memory cost by a function of the form

$$C_1(x_1) = (1 - x_1) + \beta_{1k} \; \frac{\sigma^2}{1 - x_1{}^2} \qquad (1.18)$$

where β_{1k} again requires a subjective choice.

Finally, combining a member from each of equations (1.14) and (1.17) into equation (1.11) with say memory cost twice those of a CPU ($\alpha_1 = 2\alpha_2$), one arrives at a system cost function whose level cost contours are given in Figure 1-5. While the cost function of Figure 1-5 is for but one choice of the subjective parameters β_{1k} and β_{2k}, it is still typical of the general class of such cost functions. Appendix A also contains additional parametric sets of cost contours. One significant observation which can be made from Figure 1-5 and which is generally true of other cost contours of Appendix A, is that the cost surface at its minimum

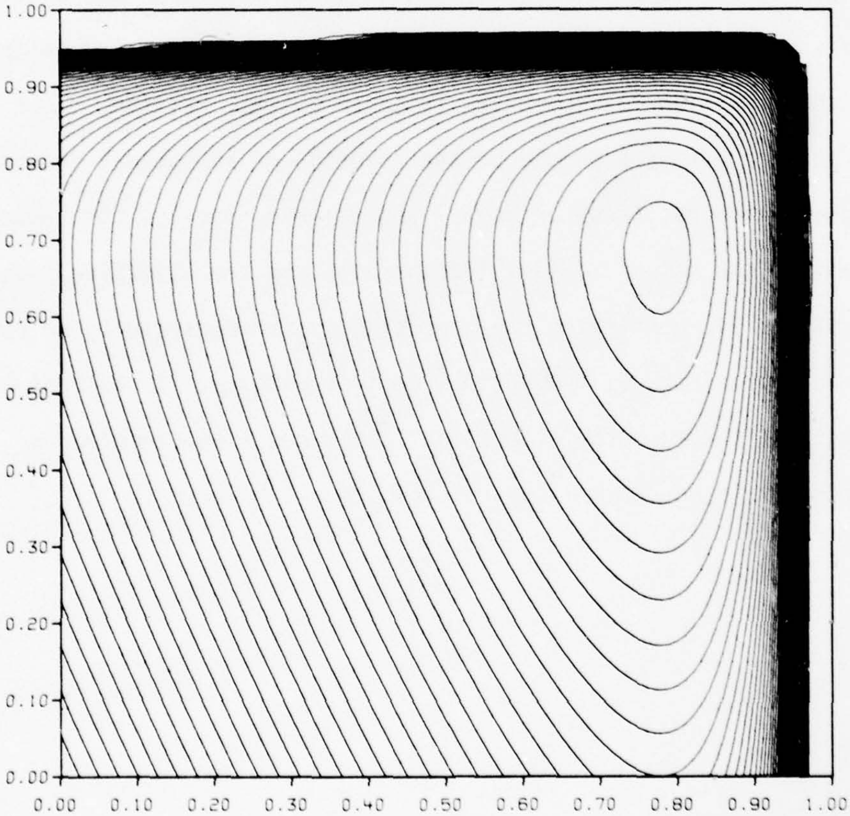


FIGURE 1-5 Operating System (Level) Cost Contours

is quite flat, i.e., has a broad minimum. For later reference, one should also note that about the desired operating point — the minimum — a quadratic function would yield a good approximation to the system's cost function.

1.5 Summary

This chapter has given an optimal control-theoretic formulation to the resource allocation problem for computer operating systems. The problem formulation was originally motivated by noting that the job scheduler has the primary mode of control upon the utilization levels of the system's resources. Once having a control problem, one should strive to accomplish it in an optimal fashion. That is, the applied controls should be the solution of an optimal control problem which has three principal ingredients, namely, the systems dynamics, the set of admissible controls, and the objective (cost) function. By identifying the operating system equivalents to each of the ingredients of the optimal control problem, it has been shown that the resource allocation (job scheduling) problem is equivalent to an optimal control problem.

CHAPTER 2

SURVEY OF RELATED RESEARCH

2.1 General Resource Allocation Formulations

The problem of allocating resources has predated computer operating systems by many years. Consequently, it is natural to first examine the applicability of allocation techniques to the resources management function of operating systems.

As a first illustration of the general allocation techniques, consider the simple one-dimensional prototype problem known as the knapsack [Dan57] or cargo-loading [BeD62] problem. Formally stated, the knapsack problem is:

> The Knapsack Problem Given the positive known integer W and the N-vectors $\underset{\sim}{v}$ and $\underset{\sim}{w}$ of positive integer values v_i and w_i, respectively, find the N-vector $\underset{\sim}{x}$ which maximize the objective function
>
> $$J(\underset{\sim}{x}) = \sum_{i=1}^{N} v_i x_i \tag{2.1}$$
>
> subject the constraints:
>
> $$x_i \geq 0 \text{ and integer (or binary)}, \tag{2.2}$$
>
> and
>
> $$\sum_{i=1}^{N} w_i x_i \leq W. \tag{2.3}$$

In words, the problem is, given the weights w_i and values v_i of eachof N items, find that subset of the items that maximizes the total value J and which satisfies a constraint on total weight W.

While the one-dimensional allocation problem has a simple and direct solution, it is well known [BeD62] that the multi-dimensional allocation processes provide formidable and challenging difficulties. Some multi-constraint integer programming problems can be reduced to a single-constraint integer program for which knapsack algorithms exist. The potential of the multidimensional methods depends crucially upon ones ability to solve the general single-constraint knapsack problem which may have negative (cost) values and bounded variables. The recent survey paper [SaK75] by Salkin and de Kluyver provides a thorough treatment of the knapsack problem. Therein, the authors note that a variety of techniques have been developed in recent years for the solution of knapsack problems. They have classified the solution methods in four categories, namely:

i) dynamic programming [BeD62]

ii) network approach [Hu69]

iii) integer programming methods including branch-and-bound and other enumerative algorithms [Hu69] [LaW66]

iv) heuristic search and Lagrangian methods [Eve63]

Some solution algorithms have features of several categories and some solution techniques are known to be equivalent to others.

For example, Shapiro [Sha68a] [Sha68b] has shown that the knapsack problem is equivalent to the shortest route problem.

In the application of allocation processes to computer systems, Denning in his thesis [Den68a] was the first to recognize the applicability of the one-dimensional knapsack problem to the memory allocation problem. Therein, he applied the well-known solution algorithm [Dan57] to the problem of selecting some "best" mix of jobs from queues of various size jobs subject to some constraint of less than full memory utilization - his so-called memory balance. Denning then goes on to formulate the joint allocation of the memory and the CPU resource as a two-dimensional knapsack problem whose solution he left as an area of future research.

In another thesis, Mahl [Mah70] extends Denning's multidimensional knapsack formulation to provide a better time synchronization between two jobs which are sharing the CPU resource. Mahl also provides a heuristic algorithm which yields approximate solutions of the multidimensional knapsack problem. His solution method being best classified under category (iv) above.

Thesen [The73] has applied a branch-and-bound technique of category (iii) above to the problem of scheduling programs in a multiprogramming environment. This approach, the author admits [The75], leads to excessive computer time and/or computer storage. Consequently, Thesen [The75] has published a recursive branch-and-bound algorithm for the multidimensional knapsack problem which is more efficient and less demanding of more storage. Its merits for computer scheduling, however, has not been demonstrated.

In a parallel processing context, Lew [Lew75a] first formulates the static one-dimensional resource allocation problem as a minimal-cost problem. He then transforms the posed discrete optimization problem into one of finding the shortest path in a graph. Then, rather than applying some of the network techniques (category ii), he applies dynamic programming methods for solution. It is not clear why he did not apply dynamic programming (category i) directly to his original formulation. Lew went onto formulate a time varying allocation problem as a sequence of static allocation problem. The resulting optimal solution given by the dynamic programming algorithm provides a computational burden far beyond any practical application.

One final area of application of allocation processes is in the pricing of computer services. It is well known (see [BeD62], p. 51) that when one uses the Lagrange multiplier technique for allocations involving costs, the multipliers λ_i (or their negatives) have the significance of a price. Kameda [Kam75] has proposed a scheme wherein such co-state variable (so-called shadow prices) are used as implicit control parameters to improve system load balance, rather than real prices charged to users. Earlier proposals on the use of pricing mechanisms to improve computer resource utilization can be found in Mahl thesis [Mah70] and papers by Nielsen [Nie68][Nie70] and Hamlet [Ham73].

As a general assessment of classical allocation techniques, one must admit that they are somewhat less than practical in their application to computer systems. Denning [Den68a] recognizes that even his static one-dimensional allocation of memory can only be optimal over short intervals. Mahl [Mah70] disavows any claim to having solved the computer scheduling problem. Thesen [The75] has noted the excessive computational requirements for his application of the branch-and-bound algorithm. Lew admits that the objectives of his paper [Lew75a] are theoretical and pedagogic rather than practical. The implementation of Kameda's scheme [Kam75] also appears to require excessive computations (or storage) or else suffer from numerical stability problems.

2.2 General Control-Theoretic Formulations

Clearly the performance of a fixed parameter resource allocation algorithm degenerates as the system's workload deviates from its set of assumed characteristics. Consequently, the operating system in its management of jobs and resources should strive to provide real-time estimation and control in order to compensate for the non-stationary nature of the offered-job-mix U_t. The obvious solution — at least for this author — was to view the computer system as any other system process requiring control [Shi67]. However, a survey of the pertinent literature in early 1974 failed to establish that computer operating systems had been given an adequate control-theoretic formulation. Subsequently, the state-space formulation presented in Chapter 1 was first developed by the author and Gagliardi and first presented in [ArG74].

Earlier, Denning in his thesis [Den68a] noted the lack of a unified treatment of memory and processor scheduling. His approach was to first introduce numerous terms such as *memory demand*, *processor demand*, *system demand*, *demand space*, *balance*, etc. His *actual demand* point $\underset{\sim}{D}(t)$ in *usage space* U is effectively equivalent to our state-vector $\underset{\sim}{x}(t)$ in state-space. Table 2-1 gives the equivalent elements of Denning's formulation to ours. Denning's proposed solution — his so-called *balance policy* — was to schedule jobs so as to minimize the componentwise distance between $\underset{\sim}{D}(t)$ (our $\underset{\sim}{x}(t)$) and some *desired demand point* (α, β).

While having many of the same elements as our approach, Denning's demand-balance theory falls short on two accounts. First, he does not associate any dynamics to the motion of his actual demand point $\underset{\sim}{D}(t)$. Denning's static or quasi-static view thus led to the knapsack problem formulation of the previous section. Consequently, his approach is more appropriate for determining very long-term or static equipment configurations [Den69] rather than the real-time resource allocation in an operating system. Secondly, Denning is not willing to interpret his usage-space (our state-space) as a metric space. Our position is that the system state-space can be metricized by the system cost function $C(\underset{\sim}{x})$. Denning's *desired demand point* (α, β) being the point $\underset{\sim}{z}$ for which $C(\underset{\sim}{x})$ is minimal. Denning's so-called "path effect" can be visualized by a system performance measure which is given by the time-path-integral of our $C(\underset{\sim}{x})$, i.e.,

DENNING	CHAPTER 1
computation: C	job: j
memory demand: $m_C(t)$	$u_{1j}(t)$
processor demand: $p_d(t)$	$u_{2j}(t)$
system demand: $\underset{\sim}{d}_C(t)$	demand vector: $\underset{\sim}{u}_j(t)$
demand space: $\underset{\sim}{V}$	offered-job-mix: U_t
usage space: $\underset{\sim}{U}$	system state-space
actual demand point: $\underset{\sim}{D}(t)$	system state-vector: $\underset{\sim}{x}(t)$
balance set: B	eligible set
balance parameters: α, β	-
-	cost functional: $C(\underset{\sim}{x})$
desired demand point: (α, β)	$\underset{\sim}{z} = \text{Min}\{C(\underset{\sim}{x})\}$

TABLE 2-1 Equivalence Between Denning's and Chapter 1 Terminology

because of jobs entering and leaving the system. Analogous to stating

$$P = \int_{\underset{\sim}{x}_0}^{\underset{\sim}{z}} C(\underset{\sim}{x}(t))\, dt. \qquad (2.4)$$

Then barring any dynamics or control constraints, the optimal trajectory for $\underset{\sim}{x}(t)$ from an arbitrary operating point $\underset{\sim}{x}_0$ to the minimum point $\underset{\sim}{z}$ is along the maximum gradient path of $C(\underset{\sim}{x})$ and <u>not</u> as Denning implies, i.e., "first balance memory, then processor." In light of the equicost contours of Figure 1-5, however, his heuristically proposed best path is certainly a practical first approximation.

In a very basic paper, Wulf [Wul69] proposed that multiprogramming systems contain a "performance monitor" whose function was to assess and, if possible, improve total system performance. As noted in Chapter 1, Wulf made most of the arguments for a state-space formulation, but failed to articulate it. Even without the explicit concept of the state of the system he did introduce estimators for many of the state-vector's components. Lacking a meaningful explicit objective function, Wulf also fails to formulate his problem as an optimization problem. He does, however, give particular attention to the dynamic problems of tuning resource allocation policies and to adjustment of workload mix.

Wilkes [Wil71] presents a control-theoretic analysis and synthesis of a "load leveller" designed by R. Mills for CTSS which provided automatic adjustment of the number of console users on the time-sharing system according to total load on the system. Specifically, the control procedure consisted of two algorithms. The first provided a prediction for the number of jobs that would be necessary for insertion into the system in order to maintain the desired load. The second algorithm controls the number of users so that the number of jobs actually entering the system is on the average equal to the predicted target number. Although of quite limited scope, the load leveller is a good example of a super-imposed control applied to a computer system.

A good part of Wilkes' paper is devoted to stability considerations which for the limited problem considered may not have been necessary for someone with a good background in predictive type systems. His stability problems are, however, quite symptomatic of complex multivariable multi-loop feedback control systems and Wilkes has elucidated a very real problem for future operating system control practitioners.

In 1972, Ashany [Ash72] noted the failure of operating system designers to avail themselves of the "tremendous potential that a state-space approach can provide." He also notes that a very important step in the analysis and synthesis of a computer system is the development of suitable models of the given "plant." Ashany, fails, however, to offer any explicit definition for the state of an operating system. Moreover, without any justification, he states that the behavior of a computer system can be described conveniently by vector matrix difference equations. As an example, he offers without discussion the linear discrete-time equivalent of our equation (1.3), namely,

$$\underset{\sim}{X}(k + 1) = A(k)\underset{\sim}{X}(k) + B(k)\underset{\sim}{U}(k). \qquad (2.5)$$

Ashany then goes on to devote the bulk of his paper to reviewing some earlier computer literature from a state-space/control-theoretic perspective.

Also in 1972, Rodriquez-Rosell [R-R72] noted that it was suprising that the techniques developed in control theory had not been systematically applied to the computer job scheduling problem. He then proposes to formulate the scheduling problem as a problem in process control. After decrying the lack of argreement upon state-variables and the futility of trying to write the system's equations, he then proposes that the problem be recast as one in adaptive control. Again, after noting the difficulties involved in choosing an objective function and a control vector, Rodriquez-Rosell goes on to review the tracking or state estimation problem. In general, his paper suffers from a poor understanding of estimation and control theory. Moreover, the author seems less than convinced that a control-theoretic approach to job scheduling would improve system performance.

Adaptive control theory has also prompted Blevins and Ramamoorthy [BlR72] to propose their so-called Dynamically Adaptive Operating System (DAOS). In their view, DAOS is functionally divided into three separate processes of, namely, Identification, Decision, and Modification. The authors introduce the terms descriptor and descriptor set rather than use state-variable and state-vector.

The purpose of the Identification process is to estimate or model the system's descriptor set. The Decision step performs status decisions and the Modification step changes system policy to better optimize performance. Blevins and Ramanoorthy present a very general formulation with quite arbitrary scope for the system descriptor set. However the author's main thrust is to provide real-time identification of random variable models for CPU and I/O service times. Implementation considerations are addressed in a series of simulation experiments directed toward the design of an adaptive CPU scheduler.

Wilkes [Wil73] also notes in 1973 the lack of dynamical models for a computer operating system. He reiterates the stability problem and notes that some knowledge of "plant dynamics" must be built into the control subsystem of any system (e.g., an operating system) which has necessarily long delays. After reviewing a long list of factors and associated policies which effect an operating system, Wilkes indentifies various control points within the system and relates them to the usual algorithms of a queueing theory approach. He is quick to realize that one particular control point — the job scheduler from the ineligible to eligible list — needs special attention in that it provides a basis for global control. His overall view that emerges is that one should provide two levels of control, namely, the global job scheduler and various subordinate local controllers.

Recent papers by Badel, et al [BGLLP74] [BGLP75] have employed adaptive control techniques to optimize the performance of virtual memory and time-sharing systems. They take a closed queueing network model as their "plant" requiring control. Their performance measure (criterion) was assumed to be a function of a single control variable, namely, the degree of multiprogramming. The author's main objective being to maximize the utilization factor of the various processors in the system with the desirable side-effect of minimizing the user's response time. A series of simulation experiments demonstrated the improvement of system performance and the proper operation of the adaptive scheme to time-varying loads.

Courtois [Cou75] has applied the "variable aggregation" technique and the concept of "nearly decomposable systems", both borrowed from Econometrics, to the study of multiprogramming systems. Variable aggregation is a technique used by economists to analyze large complex systems of many interacting state-variables. The basis of the technique is a recognition that most of the system's state-variables can be aggregated into a much smaller number of groups. The variable interaction within groups being disconnected from those in other groups and the overall system behavior is described by only the interactions among groups independent of the intra-group interactions. Such systems were qualified as being completely decomposable. When the inter-group interactions are weak compared to the (intra-group) interaction within the individual groups the system is said to be nearly completely decomposable.

For such nearly decomposable systems, Simon and Ando [SiA61] showed that short-run dynamics can be distinquished from long-run dynamics. Moreover, they showed that the strong interactions within each subsystem (group) quickly produce (i.e., short-run) a local equilibrium while the weak inter-subsystem interactions make themselves felt only in the long-run dynamics in which the whole system, if stable, will move toward a global equilibrium.

Courtois also takes a closed queueing model for the computer system upon which to apply his borrowed techniques. His contribution has provided another perspective to the modeling of the dynamic behavior of computer systems with an elucidation of the various parts and parameters which effect the system's stability [Cou75].

Gaver and Shedler [GaS71] improperly use the phrase "Control Variable Methods" in the title of their paper on the Monte Carlo simulation of a closed queueing model. Another paper by Prabhu [Pra74] titled, "Stochastic Control of Queueing Systems" would appear to be applicable to computer systems. However, Prabhu's concern is to choose an optimal stopping time to maximize profit — hardly the goal of any future-looking computer operations manager.

2.3 Models of Program and System Behavior

In a recent survey paper [DeG75], Denning and Graham note that there are two converging streams of research into the modeling of computer system behavior. One stream comprises the modeling and analysis methods of queueing theory, particularly as networks of interacting queues. The other stream comprises the study of program behavior and memory management algorithms. The authors also imply that the eventual confluence of these two converging streams of research will allow future system designers to confidently predict system behavior.

Whether such a confluence can ever be achieved is doubtful. As noted in Chapter 1, queueing models whose services times reflect CPU usage cannot simultaneously model the memory or other resource requirements. Models of program behavior, on the other hand, are primarily directed toward modeling the demands made by a program upon the system's memory resource. This author's doubts about a unification of these two modeling disciplines stems from a fundamental mismatch between their state-spaces. Queueing models which treat the CPU and other processor resources have a state-space which reflects the number of jobs and their distribution among the various queues. Program behavior models such as Denning's working-set model [Den68b] with its associated memory demand require state-variables which reflect utilization levels.

In keeping with our state-space formulation of Chapter 1, it is felt that the utilization levels for each system resource are the proper set of state-variables which can lead to a unified model for system behavior. Consequently, this thesis will emphasize the program behavior models to the near exclusion of queueing-theoretic models.

One result of the control-theoretic formulation was the recognition of the current lack of validated dynamical models for the behavior of an operating system from a resource utilization viewpoint. Moreover, past measurements upon operating systems have not been of the type required for validating the desired input-output system models. The lack of such measurements is seen as a direct consequence of the dominance of queueing models.

2.3.1 Locality and Memory Usage Models. Experimental studies of program behavior by Belady [Bel66], Liptay [Lip68], Coffman and Varian [CoV68], Hatfield [Hat72], Chu and Opderheck [ChO72], Spirn and Denning [SpD72], Rodriquez-Rosell [R-R73], and Oliver [Oli74], have repeatedly shown that for most practical programs, a program tends to favor a subset of its pages during any time interval and that the membership of the set of favored pages tends to change slowly. Attempts to formalize this empirical observed behavior by Denning [Den68c][Den70][Den72], Denning and Schwatrz

[DeS72], Coffman and Ryan [CoR72], Denning, Spirn and Savage [DSS72], Spirn and Denning [SpD72], Shedler and Tung [ShT72], Madnick [Mad73], and Denning and Graham [DeG75], have been known as the Principle of Locality. The original discoverer of this principle seems to have been lost but as Madnick observes [Mad73], "its definition has changed in time". One might add, even by one of its principle advocates (c.f., Denning [Den68c], [Den70], [Den72] and [DeG75]). In fact, an examination of all references on locality will show that very few have precisely the same definition. Some definitions are directed toward basic program behavior while others address paged memory behavior.

With the numerous attempts to formalize the notion of locality, there has been likewise numerous attempts to provide models of program behavior which provide a parameterised mechanism for generating "program-like" memory reference strings. One reason why the modeling of program behavior has found such increasing interest is because of the large expense of determining the empirical reference strings by interpretive execution of sample programs.

The random reference model (RRM) [Bel66] is a program behavior model which assumes that each page is equally likely to be referenced at any time. For a program consisting of N pages, the individual page probabilities p_k are given by

$$p_k = \frac{1}{N}, \quad k = 1, 2, \ldots, N \qquad (2.6)$$

In this case, the time between successive reference to the same page, call the inter-reference interval, is geometrically distributed. While the RRM does not produce reference strings which satisfy the locality principle, it's prime interest is as a baseline from which to compare other models which do exhibit the locality property.

A generalization of the RRM is the independent reference model (IRM) wherein the page references $r_1, r_2, \ldots, r_t, \ldots$ are assumed to be independent trials under some stationary probability distribution $\{a_1, a_2, \ldots, a_N\}$ and such that for all $t \geq 1$,

$$\text{Prob. } [r_t = k] = a_k. \tag{2.7}$$

The inter-reference intervals are again geometrically distributed and the average inter-reference interval for the k-th page is equal to $1/a_k$ [CoD73]. Results of Sp'rn and Denning [SpD72] show that the IRM is a poor approximation to observed program behavior.

The LRU-stack model (LRUM) whose rudiments were first proposed by Shemer, et al. [ShS66][ShG69] has been studied by numerous authors, especially by Mattson, et al. [MGST70], Oden, Shedler, and Tung [OdS72][ShT72], and Denning, Spirn, and Savage [DSS72][SpD72]. A least recently used (LRU) stack is a vector $\underset{\sim}{s}(t)$ whose i-th component is the i-th most recently referenced page from the address reference string up to time t, i.e., $s_1(t) = r_t$. The stack distance d(t) is the position of r_t in the previous stack $\underset{\sim}{s}(t-1)$. The LRU-stack model of program behavior is obtained by specifying some probability distributions for the stack distances. Shedler and

Tung [ShT72] use a first order Markov chain model to subscribe a probablistic structure to the distance strings d(1), d(2),..., d(t), ... Denning, et al. [DSS72] in their so-called simple LRU stack model (SLRUM) restrict the probability structure of the distance strings to be generated from independent trials from a stationary distribution $\{\beta_i\}$, i.e.,

$$\text{Prob } [d(t) = i] = \beta_i. \qquad (2.8)$$

Experimental results [SpD72] show that even the SLRUM provides a good approximation to a program's memory demands.

Denning's working-set model (WSM) [Den68b] is an extrinsic model which attempts to define locality in terms of the observable properties of the program's reference string rather than by some internal (intrinsic) generation scheme. A program's "working-set" W(t, T) at time t is defined to be the set of distinct pages referenced among the last T references, and as such is a measure of the programs locality at time t. In subsequent years, Denning, et al. [DSS72] formalized a general locality model (GLM) defined as the sequence

$$(L_1, t_1), (L_2, t_2), \ldots, (L_i, t_i), \ldots, \qquad (2.9)$$

where L_i is the i-th locality and t_i is the holding time in L_i. As such, a semi-Markov process [How64] should provide a suitable model for its underlying probabilistic structure.

Denning and his co-workers [DSS72] have also specialized the general model to their so-called simple locality model (SLM) wherein all localities L_i are assumed to be of the same size ℓ and the holding time distribution for each locality be geometric with mean holding time $1/\lambda$. At any given time t, the probability of a referencing a page in the current locality is $(1-\lambda)$ and the probability of producing a page fault being λ. The particular interval page or external locality referenced is described by two different probability distributions, say $\alpha(x)$ and $\beta(y)$, respectively. A final specialization to all uniform distributions (i.e., $\alpha(x) = \beta(y) = \frac{1}{\ell}$) yields their very simple locality model (VSLM).

Chu and Opderbeck [ChO72] provide another extrinsic model for program behavior in their considerations of their page fault frequency (PFF) policy of memory management. More recently, these authors [OpC75] have specified a renewal model (RM) for program behavior wherein the immediate reference densities of the individual pages is likened to the age-specific failure rate of renewal theory [Cox62]. When the immediate reference densities are specialized to the memory-less exponential distributions, the RM yields a continuous time equivalent to the IRM.

Lewis and Shedler [LeS73] have explored using stochastic point processes [Lew72] as models for the page exception (fault) process. The authors conclude that both the Poisson and a renewal process models should be rejected. Their best effort at modeling empirical data is provided by a two-state semi-Markov model.

Finally, Denning and Kahn [DeK75] have very recently used a semi-Markov chain to provide a macromodel for phase-transition behavior which has been observed by Madison and Batson [MaB75].

2.3.2 CPU Utilization Models. The characterization of demands made upon the CPU and other processor resources of a multiprogrammed system fall into two general categories. The first category — called the job shop models — characterizes the workload of jobs and sub-tasks as a set of triplets (i, j, k) denoting the fact that processor k must be used to perform operation j upon job i; each individual processing operation having a determistically specified processing time and possible sequencing constraints. Such models and their associated scheduling problems are treated in Chapter 3 of the text by Coffman and Denning [CoD73]. The lack of *a priori* information on job and subtask processing times coupled with the scarcity of real-time computationally feasible solution techniques for these models prevents much real interest in them for operating system applications.

The second characterization of CPU demands is as the service time distribution of various queueing models. The usual assumption of exponential service times has proven to be a crude but not unacceptable approximation to observed service times. Walter and Wallace [WaW67] indicate that a more precise fit to empirical data can be obtained by assuming that service times are hyperexponentially distributed. As noted above, Blevins and Ramamoorthy [BlR72] devote considerable effort to the modeling and estimation of resource service times.

2.3.3 Device Utiliziation Models. Often, the major factor which determines overall performance in a multiprogramming system is the rate at which information can be transferred between main memory and auxiliary storage devices. Since rotating disks and drums have been used for auxiliary storage on most past and current systems, they have received considerable attention in the literature. Consequently, our interest will also be limited to such devices.

Because drums (and fixed-head disks) require only the angular position to describe their physical configuration, they are intrinsically simpler than the large moving-head disks which require both the angular position and the position of the heads. Evidently, the analysis and properties of drum operations constitutes a subset of those for disk operations.

Treating the more general disk operation, there are three principal components, exclusive of queueing delays, of the time required to complete a transfer between auxiliary and main storage, namely: 1) the seek or head positioning time, 2) the rotational latency, and 3) the actual transfer time. Moreover, since all transfers between auxiliary devices and main memory are block tranfers, one must use buffers. Requiring buffers, the system usually requires that they be shared (pooled) and thus disk operations have a definite queueing aspect to their behavior. Consequently, disk operations have been characterized almost exclusively by queueing models. Specifically, the use of a tandom queueing model provides for the factorization of the total service time into its components of seek, latency, and transfer times. Chapter 5 of the text

[CoD73] Coffman and Denning provides a good survey of queueing-theoretic analysis of such auxiliary storage devices.

This author is unaware of any specific work which has estimated or attempted to validate models of the demand process made upon the disk or drum resource.

The interaction between both the CPU and the auxiliary devices can be treated by the queueing network models of Buzen [Buz71].

2.3.4 Parameters and Statistics of Program Behavior. While the above program behavior models require many and varied types of parameters to complete their description, only the page-fault-rate, lifetime, and working-set size statistics have received much attention in the literature. For later purposes, these three statistics are briefly reviewed below. Additional details may be found in the papers by Denning *et al*. [DeG75][DeK75].

2.3.4.1 Page-Fault-Rate Function. Denoted by $f_P(x)$, the *page-fault-rate function* gives the expected number of page faults generated per unit of virtual time when a given program reference string is processed by a memory management policy P which is subject to a main memory space constraint of x. For fixed memory space allocation schemes such as LRU, the space constraint x is interpreted to be rigid with the resident set size always less than or equal to x. In the case of variable space allocation as in a working-set policy, the parameter x is interpreted as the mean resident (working) set size.

Figure 2-1 taken from Denning and Graham paper [DeG75] shows the expected behavior of a page-fault-rate function under a LRU policy. Computationally efficient methods for determining fault-rate function exists and are treated below in conjunction with their memory management policy.

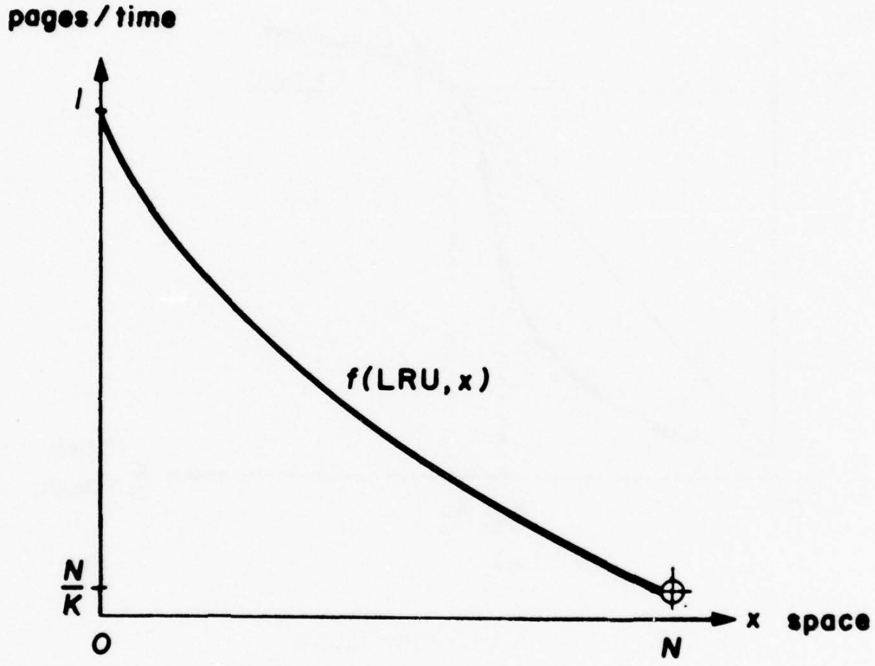


FIGURE 2-1 Typical LRU Fault-Rate Function

2.3.4.2 Lifetime Function Another measure of paged memory behavior is the lifetime function L(x) which gives the mean virtual time between page faults produced by a given policy and memory space constraint x. Specifically, it is simply defined as the inverse of the page-fault rate function f(x), i.e.,

$$L(x) = \frac{1}{f(x)} \tag{2.10}$$

The expected behavior of the lifetime function is characterized in Figure 2-2 taken from Denning and Kahn [DeK75]. It should be observed that lifetime curves are expected to have the same properties as the utility curves of Bellman and Dreyfus [BeD62].

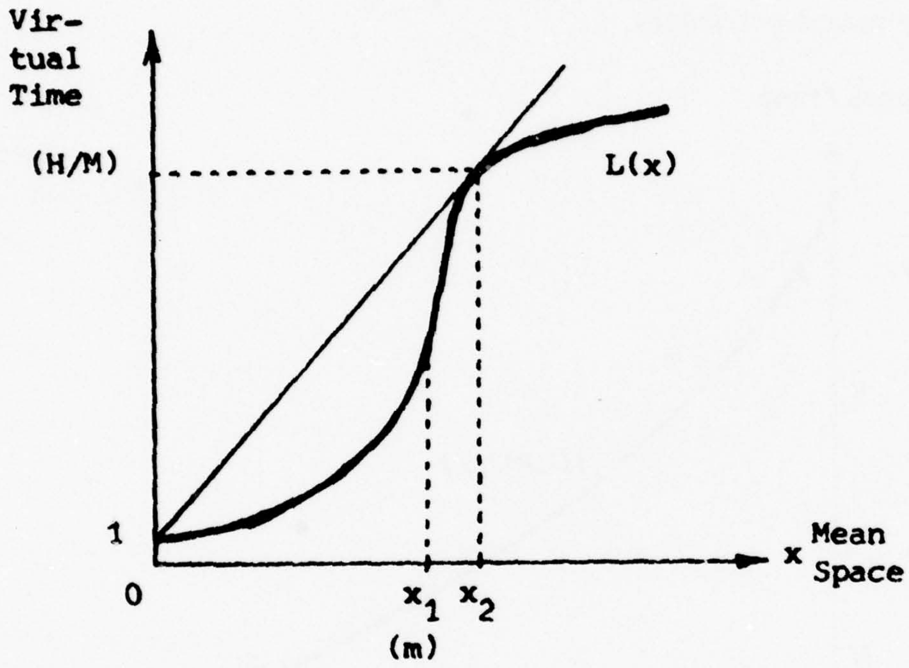


FIGURE 2-2 A Lifetime Curve

2.3.4.3 Working-Set Size. For a window size parameter T, a program's working-set W(t, T) — the collection of pages referenced during the preceding T references — has one principal statistic, namely it's size. Denoted by w(t, T), it represents the number of distinct pages which were referenced during the preceding interval of duration T. The mean, variance, and probability density function for w(t, T) are, consquently, of great interest in modeling a program's behavior. Empirical examples of w(t, T) and its related statistics are given in a recent paper by Bryant [Bry75].

2.4 Optimization Techniques and Applications

2.4.1 General Computer Resources. With the general lack of comprehensive dynamical models for computer operating systems, one should not be surprised at the complete absence of any literature on the dynamic optimization of the global performance of such systems. Even with simplifying assumptions, there are still no models which explain the complicated dependence of CPU productivity in a multiprogrammed environment to the selected memory management algorithm. Most efforts to improve system performance have made the implicit assumption that the higher utilization of main memory and the lower expenditure of time on the exchange operation between main memory and some backing-store, the higher will be CPU productivity and job through-put. Consequently, as a first approximation, the improvement of the various sequencing and allocation policies have been estimated separately yielding a piecemeal approach to dynamic optimization of computer system performance. Some of these isolated (local) improvement or optimization attempts are discussed below.

One should not infer, however, that the separated optimization schemes have been unimportant. Much to the contrary, there have been two areas which have led to sizeable improvements of system performance. The first area is the identification and removal of system bottlenecks by the analysis of queueing network models by Buzen [Buz71a]. The second area is the control of the degree of multiprogramming which, as observed by Wilkes [Wil73], provides the best basis for global control. In this direction, the best efforts have been by Gelenbe and his co-workers [GPBL73] at IRIA in France. Specifically, in Badel, et al. [BGLP75], they provide an adaptive scheme to maximize processor utilization with the degree of multiprogramming as the single control variable.

The computational complexity of the solution techniques for the multidimensional allocation processes of Section 2.1 also precludes their application to the dynamic control of all system resources. At best, they are applicable to static problems such as file allocation or to those resources which have sizeable time constants such as a moving-head disk (see Section 2.4.4 below). The techniques of the optimal allocation of resources has also been effectively applied to the problems of system configuration [Den69] which certainly impact the design of an operating system. While such considerations are beyond the scope of this dissertation, the recent paper by Mahmoud and Riordon [MaR76] nicely demonstrates the computational requirements for the optimal allocation of but two resources (files and channel capacity) in a distributed computer network. Namely, a computational burden — even for deterministic parameters — far in excess of any conceivable operating system overhead.

2.4.2 Memory Allocation and Paging Algorithms. In one of the earliest papers on virtual memory systems, Belady [Bel66] describes an optimal page replacement algorithm called MIN. MIN selects for replacement that page which will not be referenced for the longest time in the future. For a fixed number of main memory page frames, MIN causes the fewest possible number of page faults for an executing program. While MIN is unrealizable — it requires knowledge of the future portion of a page trace — it has provided a valuable benchmark for the comparison of other realizable page replacement algorithms. Belady's early paper also considers a random replacement scheme, called RAND, which is the natural counterpart of the random reference model (RRM) of program behavior. RAND too has also been valuable as a benchmark at the other extreme from MIN, (i.e., no knowledge vs. complete future knowledge).

In the ten years since Belady's paper [Bel66], a multitude of paging algorithms have been proposed and studied. Since there already exists many excellent survey papers on Memory Management by Mattson, et al. [MGST70], Denning [Den70], Parmelee, et al. [PPTH72], Aven, et al. [AKK72] and Denning and Graham [DeG75], any attempted survey here would be quite redundant. However, for later reference and comparisons, several of the more popular algorithms are discussed below.

Fortunately, for our purposes with a view toward optimal schemes, the past algorithms which are of interest can be discussed in a unified way on two accounts. First, they are all demand paging algorithms. Aho, Denning and Ullman [ADU71] have shown that demand paging is optimal if the cost C(k) of bringing in k pages simultaneously is not less than the cost of bringing them in individually (i.e., if $C(k) \geq k$). While some computer hardware would support the case when $C(k) < k$, reasonable algorithms for the management of secondary memory preclude any such cost savings. Consequently, the case $C(k) = k$ is reasonable and the restriction of future considerations to demand paging algorithms is quite realistic.

Secondly, past paging algorithms of interest can all be derived from an informal "Principle of Optimality". For any epoch t of time, define $\tau_i(t)$ to be duration of the time interval from t until the next reference to the i-th page. Then the informal principle is expressed as follows:

> Principle of Optimality [Den70] Upon a page fault, replace that page j for which $\tau_j(t)$ is maximum. If the future is not precisely known, then replace that page j for which the conditional expectation,
>
> $$E[\tau_j(t) | r_0, r_1, \ldots, r_t],$$
>
> is maximimal.

If the future of the page reference string $\{r_t\}$ is known, application of the optimality principle yields Belady's physically unrealizable MIN algorithm.

To obtain physically realizable algorithms from the principle, it is necessary to make assumptions about the probability distributions of the reference string or the quantities $\tau_i(t)$ which allow conditional expectations $E[\tau_i(t) | r_0, r_1, \ldots, r_t]$ to be computed. For example, the FIFO (first-in-first-out) algorithm results from an assumption that the inter-reference interval for all pages is equal and uniformly distributed. Using the backward interval $t-t_i'$

where $t_i' < t$ is the moment of time at which the i-th page was last referenced as an estimate for the forward reference interval $\tau_i(t)$ yields the popular LRU (least-recently-used) algorithm discussed below. Using a locality model which provides a slowly time-varying ranking of the individual page reference probabilities, say $a_i(t) < a_j(t)$, Denning's [Den70] working-set (WS) principle also follows (heuristically) from the optimality principle by the correspondingly induced rankings of the expectations, i.e., from $E[\tau_i(t)] < E[\tau_j(t)]$.

Despite its wide acceptance and intutitive appeal, the Principle of Optimality is known not to hold for arbitrary assumptions about the probabilistic structure of reference strings [ADU71]. While the informal principle is not always optimal, experience and experimental evidence suggest that the principle is a very good heuristic for reasonable assumptions about program behavior.

Motivated by the informal principle of optimality, Aho, Denning, and Ullman provided the first basic paper [ADU71] to formalize certain aspects of the memory allocation problem. Specifically, they develop formal definitions for a paging algorithm as a stochastic finite-state sequential machine. A cost function depending upon the paging algorithm, the memory state, and the program reference string is also formally defined. Finally, for an ℓ-th order Markov process model for the generation of a program reference string, the authors identify the optimal page replacement algorithm as the solution of a multi-stage

Markovian decision process. Only for zero-th order Markov generation models where the author is able to solve by dynamic programming for the optimal algorithm, denoted A_0. When specialized to the independent reference model (IRM) above, A_0 reduces to the rule "in a fixed memory allocation of m page frames, always keep the (m-1) pages of highest individual page reference probabilities a_k". For more complicated models of program behavior, the strict formalism of Aho-Denning-Ullman has been to complex to yield corresponding optimal algorithms.

Using a different cost function and a variation (ergodic) on the Markov generation model, Kogan [Kog73] was the first to recognize the applicability of Howard's "Policy-Iteration" method [How60] as a general solution technique to the multi-stage Markovian decision process of the Aho-Denning-Ullman formalization. Kogan's approach also provides a lower bound on the performance of suboptimal replacement algorithms.

Ingargiola and Korsh [InK74] modify the Mealy type stochastic machine model of [ADU71] to one of Moore type (for a comparison of Mealy vs. Moore type machines, see [Paz71]). The distinction in machine models is of little importance because of the behavioral equivalence of the two. Ingargiola and Korsh also employ the "policy-iteration" of Howard as a solution technique.

Neither Kogan [Kog73] or Ingargiola and Korsh [InK74] present any specific application of the policy-iteration method to finding new replacement algorithms. Neither is it clear that a resulting optimal policy would be practical to implement. Additional discussions of an optimal replacement algorithms is given in the recent papers by Boguslavskii and Kogan [BoK74], Aven and Kogan [AvK75], and Lew [Lew75b] [Lew75c].

2.4.2.1 The LRU Algorithm. The least-recently-used (LRU) algorithm replaces that page which has not been referenced for the longest time. As noted above, the motivation for this algorithm results from using the observed current backward page reference interval lengths as estimates for the forward (future) reference intervals. Experience has shown that the performance of this algorithm is generally quite good (most robust) over the widest class of page reference strings [Bel66] [Oli74].

Unfortunately, LRU is expensive to implement since it requires the maintenance of a dynamic list (the LRU stack) which arranges the referenced pages from top to bottom by decreasing recency of reference. For each memory reference, the corresponding page entry moves to the top of the stack and those entries which were above it are pushed down one position.

The position at which the referenced page is found in the LRU stack before being promoted to the top is called the stack distance. For a given memory allocation of x page frames, a page fault occurs at a given reference if and only if the stack distance of the reference exceeds x. These ideas form the basis of an efficient procedure for computing the fault-rate function $f_{LRU}(x)$ by counting stack distances in a reference string. Specific details for the estimation of LRU fault-rate functions are given in

Mattson, et al. [MGST70] or Denning and Graham [DeG75]. The LRU algorithm is in fact the archetype example of a large class of "stack algorithms" which enjoy various properties such as fault-rate functions which are monotone nonincreasing for every reference string, and which may be computed by a highly efficient procedure.

2.4.2.2 The Working-Set Algorithm. For a parameter T known as the window size, the working set W(t, T) of a program at virtual time t is the collection of distinct pages referenced by the program in the previous T references. Under a pure working-set memory policy (WS), a program's working-set is always kept in main memory. New pages are brought into main memory on demand. Any pages which have not been referenced during the previous T references can be replaced. The rate at which new pages enter the working set, the so-called missing-page rate m(T), is equivalent to the page-fault rate. Denoting the mean working-set size by $\bar{w}(T)$, the working-set fault-rate function given parametrically be setting

$$f_{ws}(\bar{w}(T)) = m(T) \qquad (2.11)$$

for T = 0, 1, 2, ... yields a direct relation between paging rate and the average space allocation.

Denning and Graham [DeG75] provide algorithms for computing all the functions m(T), $\bar{w}(T)$, and $f_{ws}(x)$. Their computational procedure for the missing page rate function m(T), however, requires a set of counter to record the occurrences of backward distances to the last prior reference of the currently referenced page. For long reference strings (several million references) the implementation of a separate counter for every possible backward distance is computationally prohibitive. While Denning and Graham do not speak to this problem, the only obvious solution is to provide some cruder quantization of the possible values of backward distances. Since many successive page references are to the same or very recently referenced pages, a uniform quantization of possible distance values would wash-out this short-term behavior. For computations of some empirical results presented later in this thesis, we have chosen to quantize the possible backward distance values into a set of values which are uniformily spaced on a log-base-two scale. Our motivation being a desire to preserve both the short term and the long-term structure of any program page trace. The non-uniform quantization in distance does, however, require modification of Denning and Graham recursion formula for calculating the mean working-set size. We have used a straight forward numerical integration scheme to derive $\bar{w}(T)$ from $m(T)$.

Rodriquez-Rosell and Dupuy [R-RD73] have addressed the design, implementation,and evaluation of the working-set algorithm in an operating system environment. The optimal choice for one of the design parameters — the window size T — has been considered by Henderson and Rodriquez-Rosell [HR-R74].

2.4.2 CPU Scheduling Algorithms. The ideal objective of all scheduling in a computer system is to minimize both the total cost of computer services and the users waiting time. In practice, these two conflictive goals have often been approached separately. The classical batch-processing system is an example which minimizes service cost with a total neglect of user response time. On the other hand, a good interactive time-sharing system responds instantly to users at considerable inefficiencies of system hardware and their costs. In terms of CPU scheduling, this dicotomy of overall system goals has prompted the application of two general basic classes of scheduling disciplines.

In batch-processing systems, task and CPU schedulers strive for efficient utilization of computer resources in order to reduce service costs. For such systems, the deterministic scheduling problem and their associated algorithms are treated in Chapter 3 of the Coffman-Denning text [CoD73]. In interactive systems, the performance criteria becomes the users average waiting time and the scheduling disciplines for the stochastic queueing models are appropriate. These too are well treated by Coffman and Denning (as Chapter 4). Very recently, there has been published two whole texts primarily devoted to computer scheduling problems. The first by Coffman [Cof76] emphasizes the deterministic or job-shop like scheduling. Kleinrock's second volume [Kle76] is devoted to the stochastic queueing models.

Most modern operating systems are, however, neither purely batch nor purely interactive, but rather provide some mix of both types of service. The selection of task and CPU scheduling should thus idealy reflect some compromise between the two classes of users and their respective type of scheduling discipline. Unfortunately, no unifying theory exists for designing (sans optimizing) such schedulers. Also, as noted above, since deterministic scheduling algorithms requires prior knowledge of job demands plus considerable computational complexity, they have not been appropriate for scheduling the CPU (or other fast response devices). Consequently, most operating system CPU schedulers are currently designed and analyzed from a queueing theoretic basis.

The queueing models for large multiprogrammed systems have, however, address the fact that the system must serve several types of users. With an overall strategy which attempts to favor users having a short CPU processing requirement, most CPU schedulers use some form of time division multiplexing of the CPU between user jobs. Thus, in turn, the scheduler allocates the CPU to a user for at most q consecutive units of time (q is usually called the quantum). Each time a user's quantum terminates, the job is returned to some queue if its total processing requirement has not been completed. The optimization of one of the design parameters of such so-called feedback schedulers, namely the quantum size q, has received some attention. Rasch [Ras70] has considered the determination of the optimum quantum length for a round-robin scheduled queue. Very recently, Potier, et al. [PGL76] have considered the improvement of user waiting time by adaptively varying the CPU processing quantum as a function of job traffic conditions.

2.2.4 Device Scheduling Algorithms. Since the use of disks and drums implies queueing processes, their associated schedulers have often used the same types of scheduling policies as has been used for the CPU. Especially for drums, the first-come-first-served (FCFS) and the shortest-access-time-first (SATF) service policies correspond to the FIFO and shortest-processing-time-first (SPTF) rules for CPU scheduling, respectively.

Disks, on the other hand, have an additional component — the seek time — to their total service time. Consequently, for disks, one frequently studied problem concerns the scheduling of transfer requests in a way that reduces disk head movement. Again, the simplest policy is to service all requests on a FCFS basis with no consideration of the resulting head movement. A second policy, termed shortest-seek-time-first (SSTF) by Denning [Den67], corresponds to the SATF policy in drums in that the heads are moved as little as possible on each seek. This is essentially a step-by-step or local minimization process. Frank [Fra69] considers the global minimization problem of finding the seek pattern which minimizes the total amount of seek time (MTST) necessary to service all requests present in the disk queue at a given time.

During periods of heavy load, both the SSTF and MTST minimization policies have the potential disadvantage of creating excessively long delays for certain requests. This is because minimizing head movements tends to keep the heads in a particular region for a long period of time. Newly arriving requests directed towards this region will receive good service since they will require comparatively short seeks. However, requests directed towards more distant regions will continue to wait since it would be sub-optimal to move the heads a large distance to serve these requests and then to move the heads all the way back to serve the newly arrived requests.

To avoid the possibility of these excessive delays, Denning [Den67] has proposed a head movement policy called SCAN in which the heads continually sweep back and forth across the disk, servicing requests for each cylinder they pass but never changing direction until the end of the sweep. Denning compares this policy with FCFS and SSTF and concludes that SCAN is the most desirable even though SSTF is more efficient.

Fuller [Ful72] considers the problem of optimally scheduling a set of N drum requests so as to minimize-total-processing-time (MTPT). He recognizes the problem to be a special case of the traveling salesman problem of deterministic scheduling theory. Fuller goes on to develop a solution algorithm which has the attractive property of exhibiting a computational complexity on the order of N log (N). Similar type algorithms should be even more appropriate for the longer service time disk devices.

Additional details and references for both disk and drum scheduling can be found in Chapter 5 of the Coffman-Denning text [CoD73] and the recent monograph [Ful75] by Fuller.

One final area of system optimization related to secondary memory devices such as disk and drums is that of file allocation. By organizing one's data so that the most frequently referenced records are on the middle cylinders, disk seek times will be reduced since the heads will never have to move more than half of a seek distance to reach such records. The possible effects of such a policy on performance has been considered by Abate, et al. [ADW68] and Frank [Fra69].

More formally, Buzen [Buz73] considered the problem of file allocation amongst drums of various access times. Such placement or balancing of I/O requests allowed him to minimize total mean response time. Likewise, Chen [Che73] considered optimal file allocation in multi-level storage systems for several performance measures including minimal response time and minimal storage costs.

CHAPTER 3

FIRST ORDER STATISTICS OF PROGRAM BEHAVIOR

3.1 Shortcomings of Current Empirical Research

While there is a wealth of both program behavior models and memory management algorithms, there is a dearth of published results comparing the various models and algorithms on empirical data.

Early papers by Belady, Coffman and Varian, and Hatfield were primarily concerned with the effects of varying page size. Belady [Bel66] compares the efficiency as a function of page size of six different (but not LRU) algorithms against his non-realizable optimal MIN algorithm. Coffman and Varian [CoV68] compare the LRU algorithm to Belady's MIN algorithm using a normalized page-fault ratio as a measure of performance. Hatfield [Hat72] uses page-fault counts for the FIFO, LRU, and "Used-bit" algorithms in his consideration of page size. None of these studies were conclusive as to which algorithm performed best on empirical page traces.

Denning, Spirn, and Savage [DSS72][SpD72] have specialized the parameters of several locality models in an attempt to approximate the observed behavior of several real programs. The model fitting was attempted to the measured average working-set size and missing-page-probability curves. They conclude that of the models considered, the LRU stack model provides the best fit.

Oliver, Chu, and Opderbeck [OCO72] report working-set measurement results for three different programs. Each program trace contained one million references and the estimated statistics include the inter-page fault time distribution, average page-fault frequency and the mean working-set size. Subsequently, Chu and Opderbeck in a series of papers [ChO72][OpC74][ChO74] presents results for their page-fault-frequency (PFF) replacement algorithm over four programs whose trace lengths were in the range of from two to five millions references. For a fixed page-fault-rate and a memory space-time product as a measure, the authors compare their PFF algorithm to the LRU and working-set (WS) algorithms. They conclude that the PFF algorithm's performance is better than that for the best (fixed memory) LRU algorithm and comparable to that of the WS algorithm.

Rodriguez-Rosell [R-R73] also presents a collection of empirical working-set model statistics including the page-fault-rate and mean working-set size curves as functions of the window size parameter T. Unfortunately, results for only a single program trace are presented.

An attempt by Lewis and Shedler [LeS73] to fit micromodels to the generation process for page-fault sequences must also be considered inconclusive.

Using the number of page faults as a measure, Oliver [Oli74] has favorably compared a Global LRU algorithm to Local LRU algorithms with either fixed or varying memory partitions. His studies used synthetic job-mixes of real programs which contained both identical and varied tasks. Oliver identifies the local LRU with varying partition as equivalent to past implementations of the working-set algorithms.

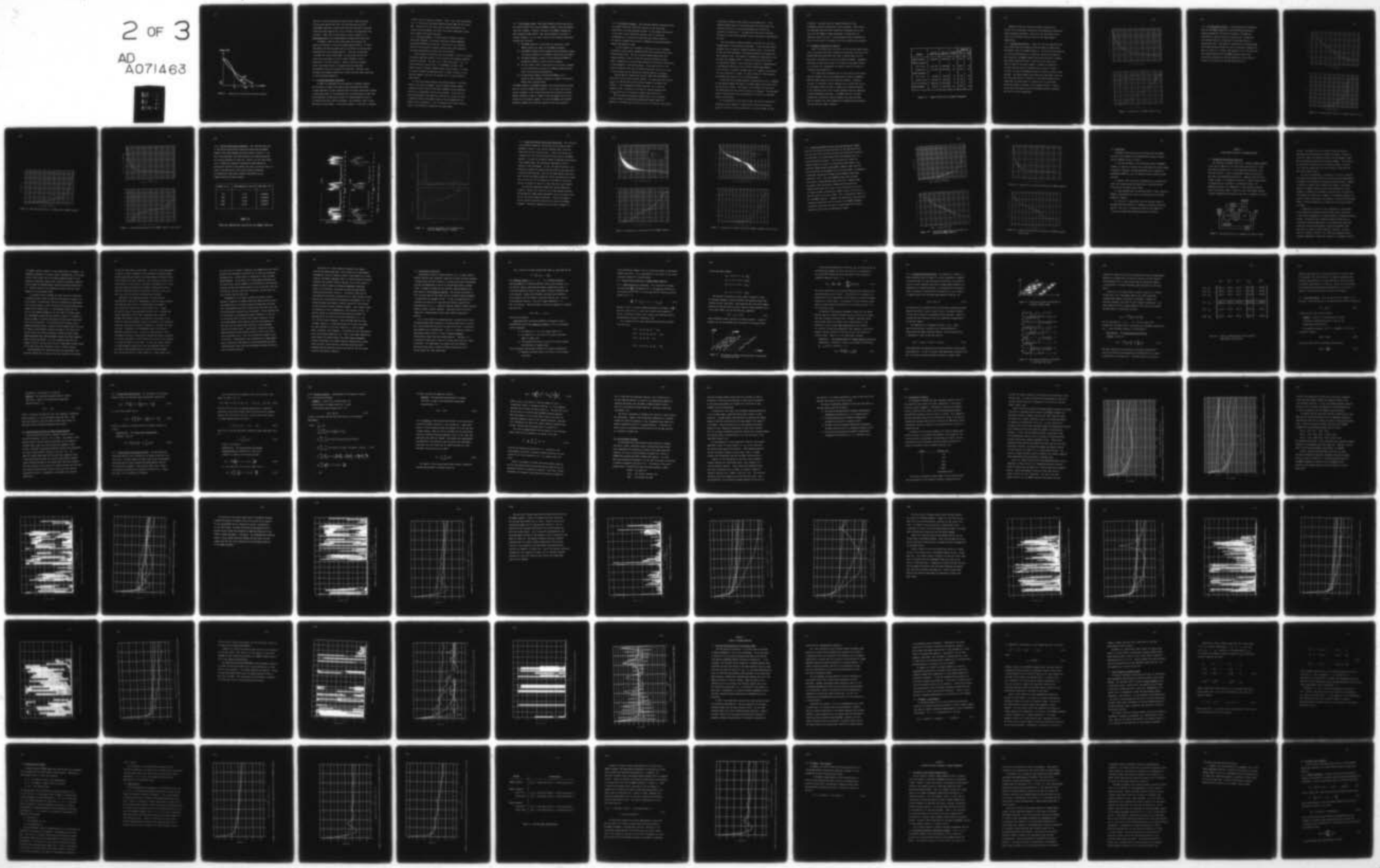
Figure 3-1 taken from the survey paper [DeG75] by Denning and Graham has been offered by the authors as a typical comparison of the LRU and working-set (WS) page-fault-rate functions. The implication is that for smaller memory allocations (less than some x_0), LRU is better than WS while for larger allocations, WS is better than LRU. Denning and Graham do not give specific empirical results but do quote unpublished work of Prieve and Fabry [PrF73] showing that WS can be as much as 30 percent better than LRU.

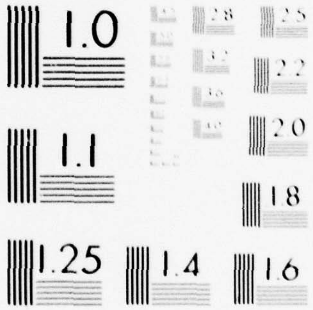
An examination of the Prieve-Fabry work [PrF73], however, indicates that their comparison between LRU and WS is not a true empirical comparison. The empirical data collected by Prieve and Fabry consisted of their so called "interval reference sets" consisting of the set of pages referenced during each period of program execution. With the collapsing of the page reference strings into their collected interval reference sets, they subsequently had to use some (random permutation) model in order to generate pseudo page trace strings for comparison of LRU and Belady's MIN algorithm with the working-set scheme.

A recent brief paper by Prieve and Fabry [PrF75] also presents

AD-A071 463 NAVAL UNDERWATER SYSTEMS CENTER NEW LONDON CT NEW LO--ETC F/G 9/2
OPTIMIZATION OF COMPUTER OPERATING SYSTEMS.(U)
APR 79 C R ARNOLD
UNCLASSIFIED NUSC-TR-6045 NL

2 OF 3
AD
A071463





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

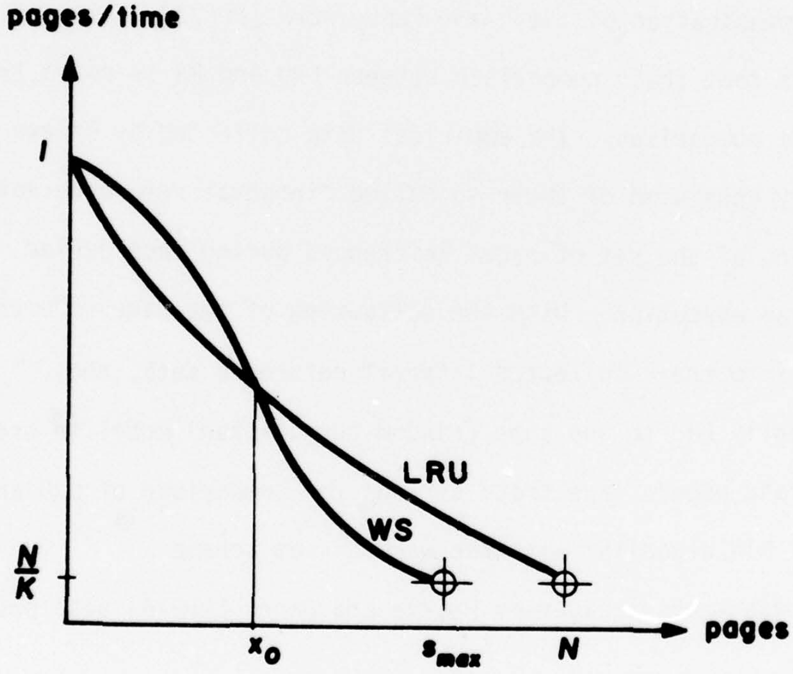


FIGURE 3-1 Comparison of LRU and WS Fault-Rate Functions

one set of results extracted from their earlier unpublished paper [PrF73] which implies that their so-called page partition (PP) replacement algorithm is again better than the working-set algorithm. Prieve and Fabry comparision is also in terms of the page-fault-rate functions. Their view of the function's graph is, however, as a performance curve in the average-memory-size/page-fault-rate plane.

In summary, there are only three papers which address the empirical validation of intrinsic program behavior models. Of those considered, the LRU stack model must be considered to yield the best approximation to real program behavior. Since the LRU algorithm is optimal for the LRU stack model, its performance against the replacement algorithms of extrinsic models [DSSp72], such as working-set is of great interest. However, there are disparate and/or inconsistent results in comparing the LRU algorithm's performance to that of the working-set algorithm. Moreover, there has been only piecemeal comparisons of other extrinsic model algorithms to LRU and amongst themselves.

3.2 A Program Measurement Experiment

In order to investigate the true behavior of computer programs and the merits of some of the popular page replacement algorithms, a major experiment has been conducted upon five actual operational programs. A second major goal for the experiment was to provide empirical data from which additional second-order statistics of program behavior such as correlation functions could be estimated. The experiment itself is best described in three phases: 1) the trace program, 2) the data or programs traced, and 3) the analysis programs. Phase 1 and 2 were accomplished by S. W. Galley upon the Dynamic Modeling System (DMS) of MIT Project MAC. The analysis of the trace data has been performed by the author using the computer facilities of the Naval Underwater Systems Center (NUSC), New London Laboratory.

3.2.1 The Trace Program. To obtain memory reference strings of operational programs, a previously developed software debugging system [Gal71][GaG74] was modified. Specifically, the simulator portion of the Execution Simulator and Presenter (ESP) system was extracted and modified to develop the Trace Program.

The environment provided by the Trace Program is briefly as follows. The Dynamic Modeling System is a DEC system-10 with hardware-supported paging and swapping. The page size is 1024 36-bit words. The time-sharing operating system in use, ITS [EGHKN69], provides a full address space of 256 pages (262144 words) to each process. Of these, the subject program uses whatever subset it needs, typically in two separate segments, and the Trace Program lives in three otherwise unused pages.

As the Trace Program executes the subject program, it collects, buffers, and writes on magnetic tape the complete memory-address reference string, except for the lowest-numbered sixteen addresses, which act as accumulators and are never swapped. Service and I/O routines provided by the operating system are not traced, but calls to them are noted in the output string so that accompanying time delays can be accounted for. For full generality of later analysis, there is no collapsing of addresses into page numbers.

3.2.2 The Programs Traced. Many Dynamic Modeling System application and system programs are written in MUDDLE [Pfi69], a LISP-like [Moo74] high-level language. Recently, the name of the MUDDLE language has been shortened to MDL [GaP75]. Most other programs are written in assembly language. For our study, traces of the following operational programs were selected:

(1) the MUDDLE Compiler in the process of converting a small MUDDLE program into input for the MUDDLE Assembler;

(2) the MUDDLE Assembler which assembled LAP-like [Moo74] programs into binary programs for the MUDDLE interpretive environment;

(3) the MIDAS Assembler (similar to the standard DEC MACRO-10 Assembler) assembling several utility programs;

(4) the Text Editor, TECO, in the process of using macro commands to walk through the structures of a MUDDLE program checking that it was correctly parenthesized.

(5) an Application Program, called VECTOR-SMOOTH, in it execution which averaged a sequence of random floating-point numbers over a moving window.

The MUDDLE Compiler, the MUDDLE Assembler, and the Application Program were all written in MUDDLE and compiled. In all cases, the input or control parameters of the traced executions were selected to provide address reference strings of length 1-5 million address over the full execution of the subject program. For both the MUDDLE and the MIDAS Assembler programs, two different execution traces were obtained.

3.2.3 The Analysis Programs. Three separate programs have been written to provide first-order statistical analysis of the trace tapes. Specifically, the three programs provided 1) the complete LRU analysis, 2) the ideal or pure working-set analysis, and 3) a practical implementation of a working-set algorithm with related statistics. All analysis programs were written in FORTRAN and executed upon NUSC's UNIVAC-1108 computer system.

In their analysis of the address reference string, all programs contained an input control parameter PGSIZE which provided for possible different page sizes. Two sets of memory address bounds were also always provided as input to allow the individual programs to check that all addresses on the trace tape properly fell within the instructions or data segments of the traced programs. Minor modification of the analysis programs could thus also provide individual results for either or both the data and the instruction fetching behavior of programs.

The LRU Analysis Program uses the well known algorithm [MGST70] which uses a set of stack distance counter. The page fault-rate function $f_{LRU}(x)$ is then derived as the fractional number of distances that exceed x. The corresponding lifetime functions L(x) is given by equation (2.10). Execution of the analysis program produces plots of $f_{LRU}(x)$ and L(x) as functions of memory space allocation x.

The Pure Working-Set Analysis Program uses a set of backward distance counters as described by Denning and Graham [DeG75] but modified as indicated earlier (Section 2.4.2.2) to provide a non-uniform (logarithmic) spacing of the values for the window size T. This program produces plots of the missing page rate function m(T), the lifetime function L(T), and the mean working-set size $\bar{w}(T)$ as functions of window size T. The page-fault-rate and the lifetime functions are also plotted parametrically as functions of mean working-set size.

The third analysis program provides a simulation of a practical implementation of the working-set algorithm. It uses an input control parameter ISIZE to specify a window (interval) size T. On the basis of this and other parameters, the program sequentially processes the input address string. Each address is mapped into a page number using the page size parameter PGSIZE. Executive function and I/O calls are counted but ignored. Each valid page reference is checked against a continuously maintained list of current resident page numbers. If not found as a currently resident page, it is added to the resident list and a page-fault is noted. For each interval of ISIZE addresses, the program also builds a list of referenced pages.

At the end of each window interval, the working-set size is computed as the average between the number of resident pages at the start and at the end of the interval. Then pages in the resident list which were not referenced during the current interval are released. The individual values of working-set size are accumulated in an array for subsequent analysis and plotting.

At the termination of the trace string, the Practical Working-Set Simulation Program computes a single value of fault-rate function $f_{ws}(T)$ as the ratio of total page-faults to the total number of page references. The mean value and standard deviation of the accumulated values of working-set size are computed. These results are tabulated along with other supporting information such as the totals for the number of pages referenced, of page faults, of intervals, and of I/O calls. The accumulated values of working-set size also are saved for subsequent statistical analysis and plotting.

3.3 A Summary Presentation of Results

Table 3-1 provides gross statistics for the various sample program traces. One should note that although the actual page size used by the Dynamic Modeling System PDP-10 was 1024, a smaller page size has been used in the analysis of two of the smaller programs. Reasonable variations of page size has had little effect on the general nature of our results so, consequently, no specific results are given on variations due to page size.

It is impossible to present here all the results of the various analysis programs with various combinations of control parameters upon the seven traces over five different programs. Therefore, our goal is to present a fairly complete set of results for but one of the program traces and then to summarize the observed behavior of the remaining traces with a primary emphasis upon the comparison of page replacement algorithms. The MUDDLE Compiler has been selected for illustrative purposes because the results derived from its address trace seem somewhat more complete (and interesting) than for the other sample programs.

PROGRAM	NO. OF INSTRUCTIONS	NO. OF ADDRESSES	# ACTUAL PAGES	ANALYSIS	
				PAGE SIZE	# OF PAGES
MUDDLE Compiler	1,904,024	2,895,872	189	1024	189
MUDDLE Assembler					
First Trace	1,227,610	1,826,086	86	1024	86
Second Trace	3,017,658	4,481,024	138	1024	138
MIDAS Assembler					
First Trace	614,698	935,936	19	128	152
Second Trace	2,144,290	3,322,738	19	128	152
TECO Text Editor	2,409,764	4,343,820	18	256	72
VECTOR-SMOOTH	839,793	1,290,240	221	1024	221

TABLE 3-1 GROSS STATISTICS OF SAMPLE PROGRAMS

Appendix B provides a much more extensive tabulation of results for the individual traces and of the comparisons between the page replacement algorithms. There also, the consistency of our results for different execution traces of the same program is demonstrated.

3.3.1 LRU Algorithm Analysis. Figure 3-2 gives the page-fault-rate and lifetime functions produced by the LRU Analysis Program from the address trace of the MUDDLE Compiler. The LRU fault-rate function is strictly nonincreasing as it should be. While not necessarily being atypical, the MUDDLE Compiler fault-rate function is somewhat more convex toward the origin — even on the log-linear plot — than for those of the other programs. Likewise, the Compiler's lifetime functions does not exhibit quite as large of a sharp increase as do the corresponding curves for some of the other traced programs. One should observe that without the logarithmic scale for the fault-rate function in Figure 3-2, its behavior would be lost except near the origin which is certainly not its significant allocation regime as observed in the lifetime function. Finally, one may note that under execution, the MUDDLE Compiler referenced only 163 of its 189 loaded pages.

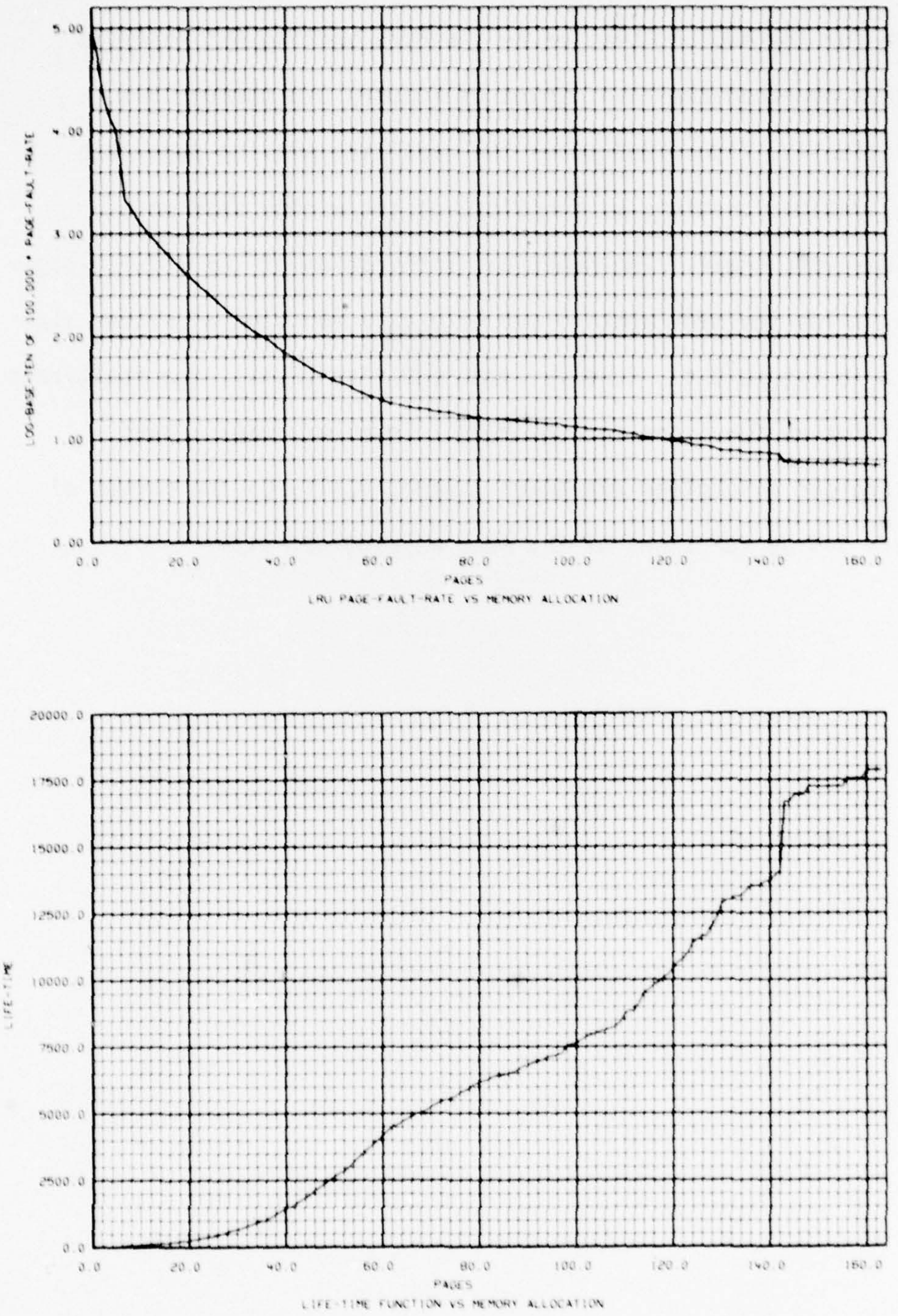
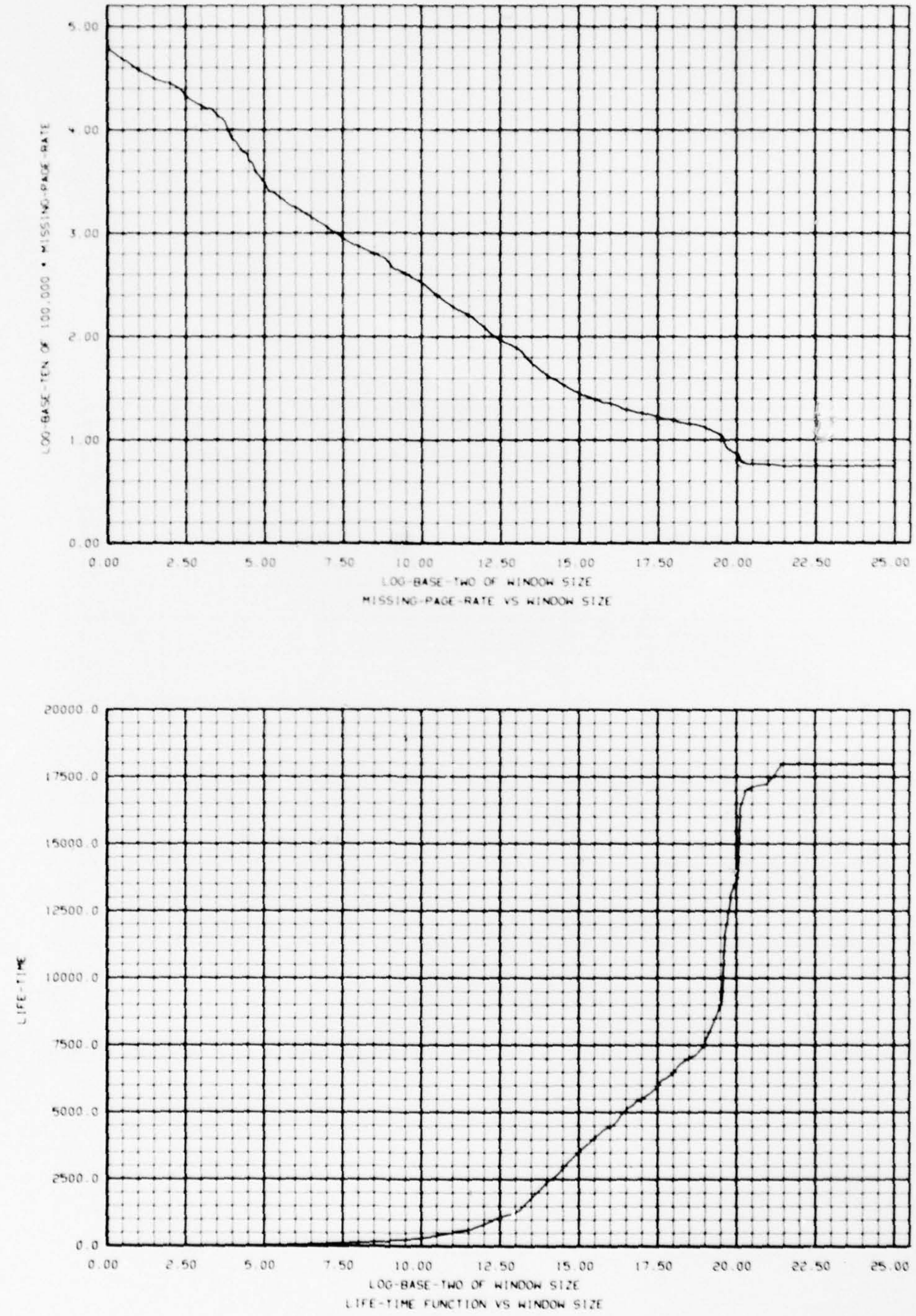


FIGURE 3-2 LRU Analysis for MUDDLE Compiler Trace

3.3.2 Pure Working-Set Analysis. The reprocessing of the address trace for the MUDDLE Compiler by the Pure Working Set Analysis Program produced the five curves of Figures 3-3 through 3-5. Specifically, Figure 3-3 gives the resulting missing-page-rate and lifetime functions as functions of the window size on a log-base-two scale. The numerical integration of the missing-page-rate yields the mean working-set size as a function of window size given in Figure 3-4. Finally, from equation (2.11), the fault-rate and lifetime functions for the MUDDLE Compiler under a pure working-set allocation are given parametrically by the curves of Figure 3-5 as functions of the mean working-set size.



MISSING-PAGE-RATE VS WINDOW SIZE

LIFE-TIME FUNCTION VS WINDOW SIZE

FIGURE 3-3 Pure Working-Set Analysis for MUDDLE Compiler Trace

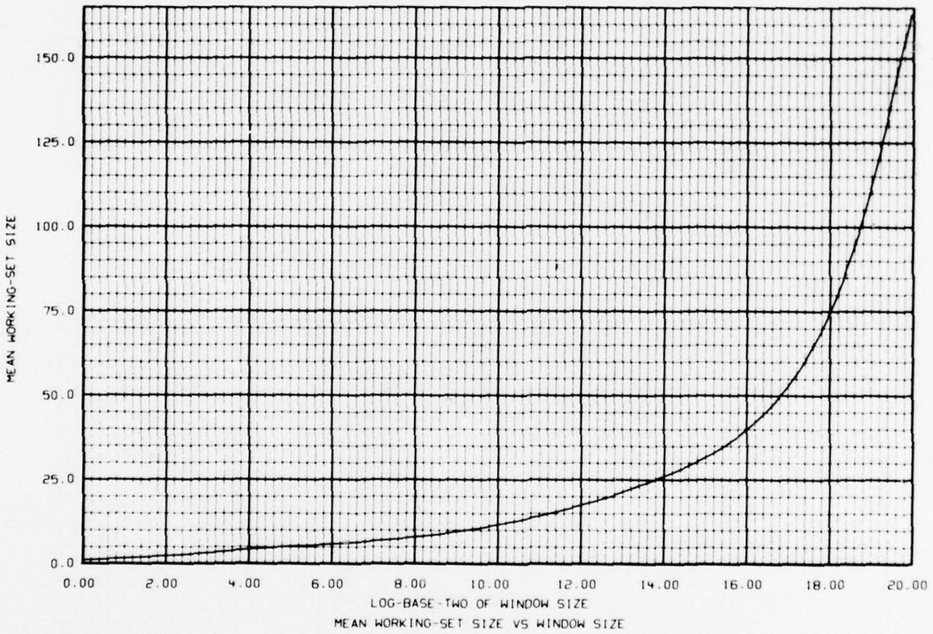


FIGURE 3-4 Mean Working-Set Size vs. Window Size for MUDDLE Compiler

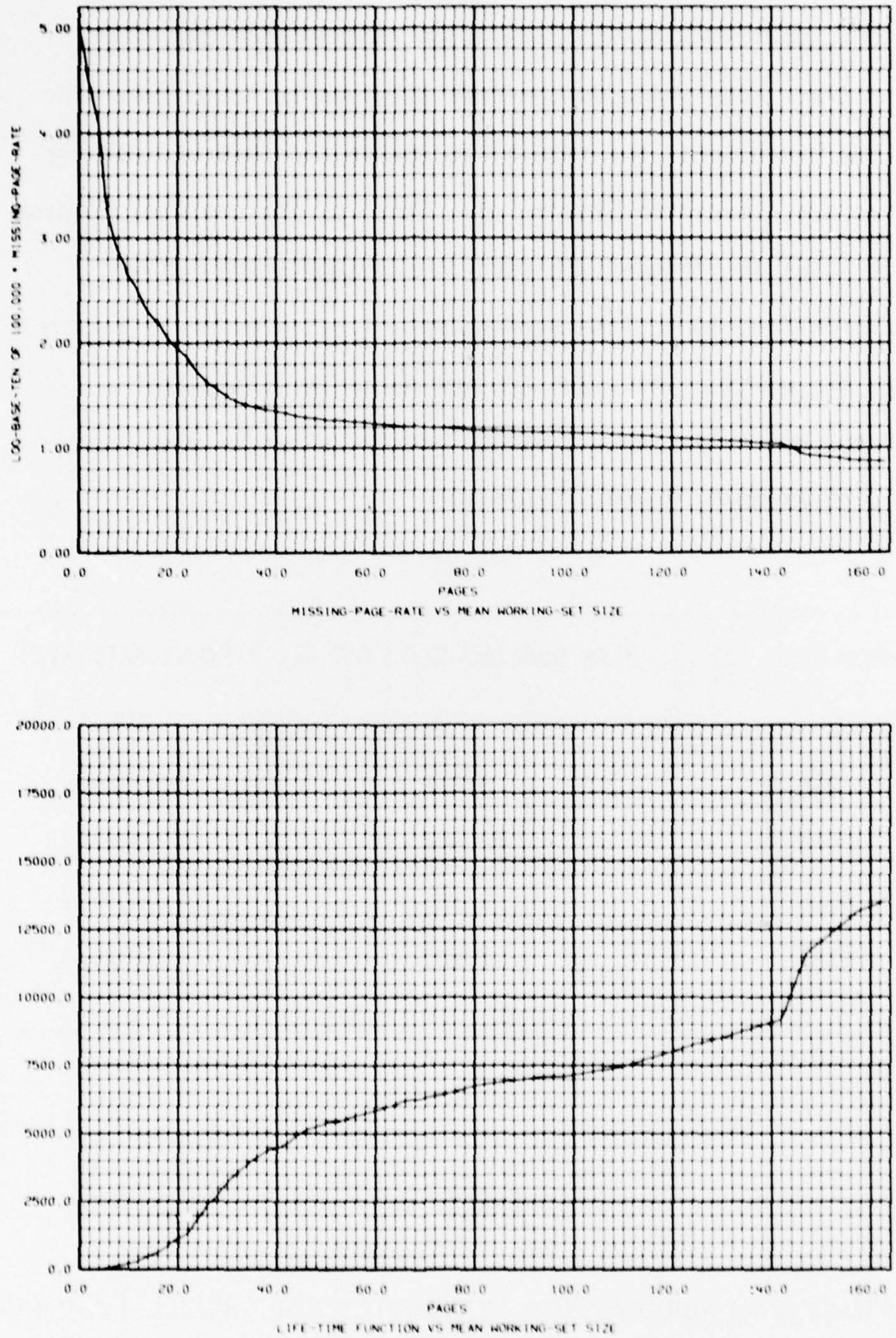


FIGURE 3-5 Pure Working-Set Analysis for MUDDLE Compiler Trace (cont.)

3.3.3 Practical Working-Set Simulation. Four separate executions of the Practical Working-Set Simulation Program upon the MUDDLE Compiler trace tape produced the results given in Table 3-2. For two of the executions, the time histories of the observed working-set sizes are plotted in Figure 3-6. Finally, for one case (window size T = 4096) the frequency histogram and sample cumulative distribution function of the working set sizes is given by Figure 3-7. Clearly, the working-set sizes are not normally (Gaussian) distributed but rather have a bimodal distribution as was commonly observed by Bryant [Bry75].

WINDOW SIZE (T)	MEAN WORKING-SET SIZE ($\bar{w}$)	FAULT-RATE f(T)
256	8.777	0.0062995
1024	12.944	0.0025742
4096	19.361	0.0009619
16384	28.591	0.0003373

TABLE 3-2

PRACTICAL WORKING-SET STATISTICS FOR MUDDLE COMPILER

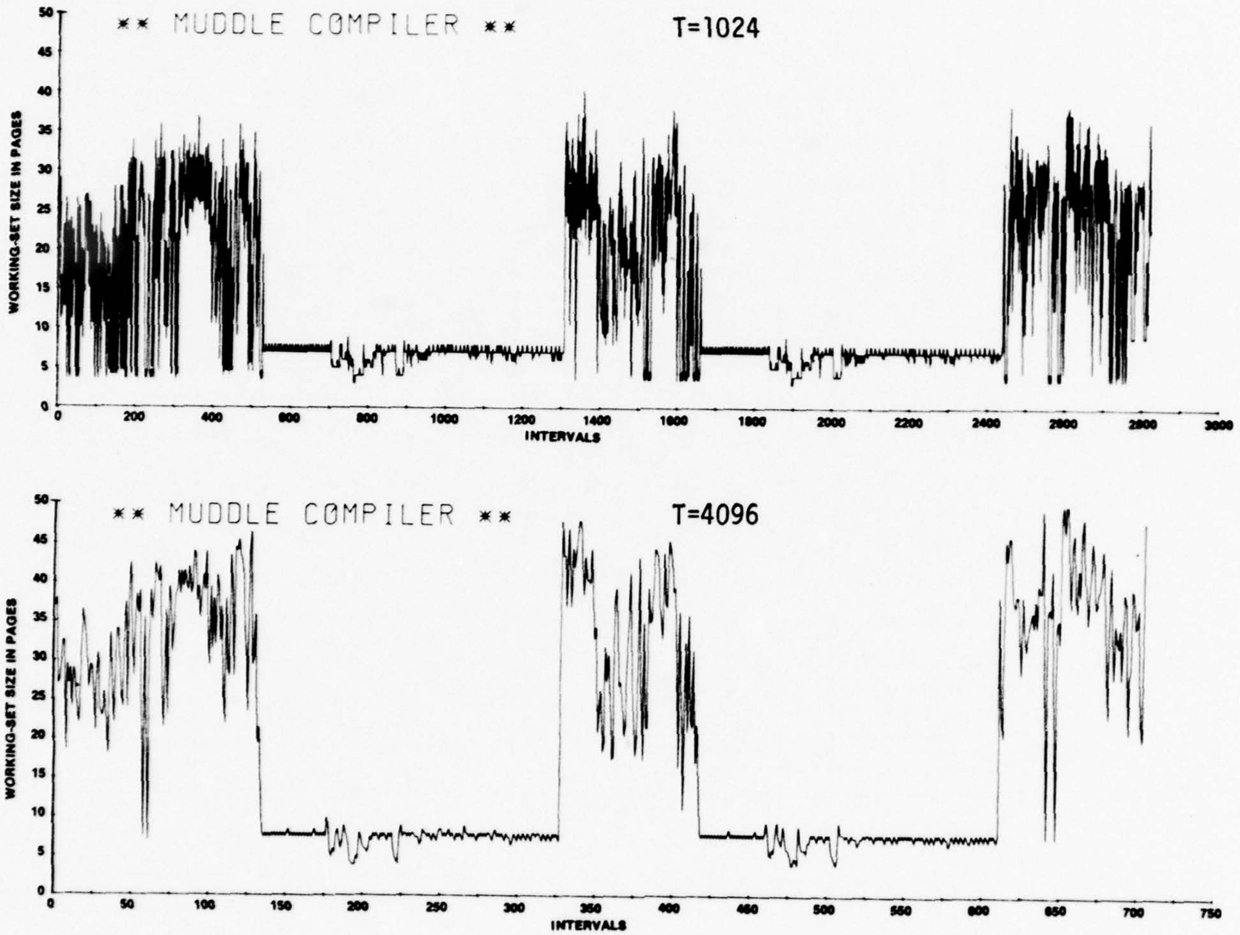


FIGURE 3-6 Working-Set Size vs. Time for MUDDLE Compiler

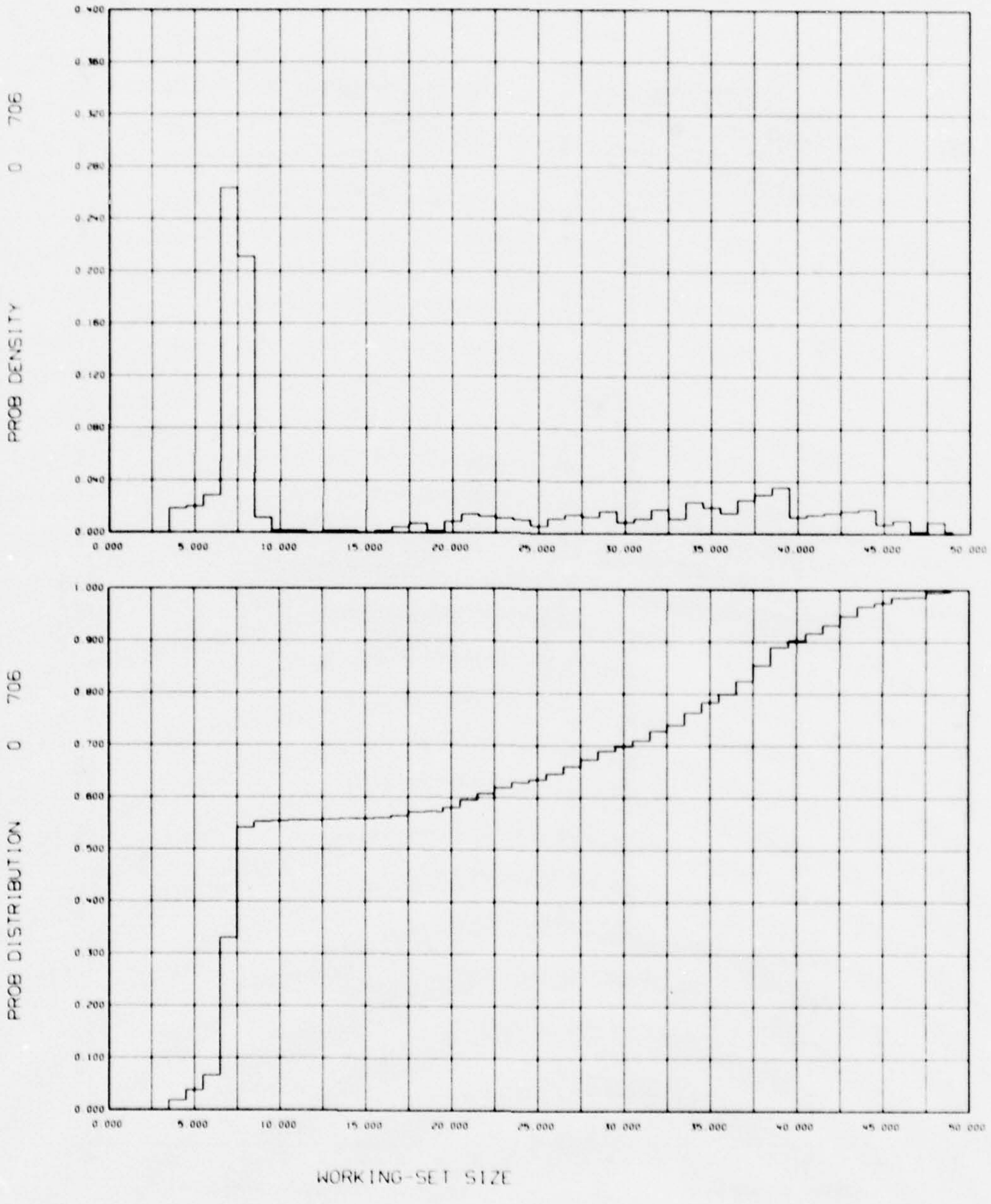


FIGURE 3-7 Histogram and Sample of CDF of Working-Set Sizes for MUDDLE Compiler (T=4096)

3.3.4 Comparison Between LRU and Pure Working-Set. The difference in performance between the LRU and the pure working-set memory management schemes is best seen by comparing their respective page-fault-rate and lifetime curves. Figure 3-8 presents such a comparison of the results derived from the trace of the MUDDLE Compiler. For most of the smaller memory allocations corresponding to the shaded region, the working-set algorithm is clearly superior to LRU replacement. In fact, for some allocations, the working-set scheme will produce only one-fifth as many page-faults as does the LRU algorithm. Only for the large memory allocations does the LRU replacement scheme out perform the working-set scheme. These results are just the opposite of the behavior implied by Denning and Graham in Figure 3-1 taken from their paper [DeG75].

The resulting comparison between the LRU and working-set schemes is typical too in that for all of the programs analyzed, the working-set scheme always performed as well as or better than LRU for the smaller memory allocations. Figure 3-9 presents a similar comparison derived from the first trace of the MUDDLE Assembler. Additional comparison can be found in Appendix B.

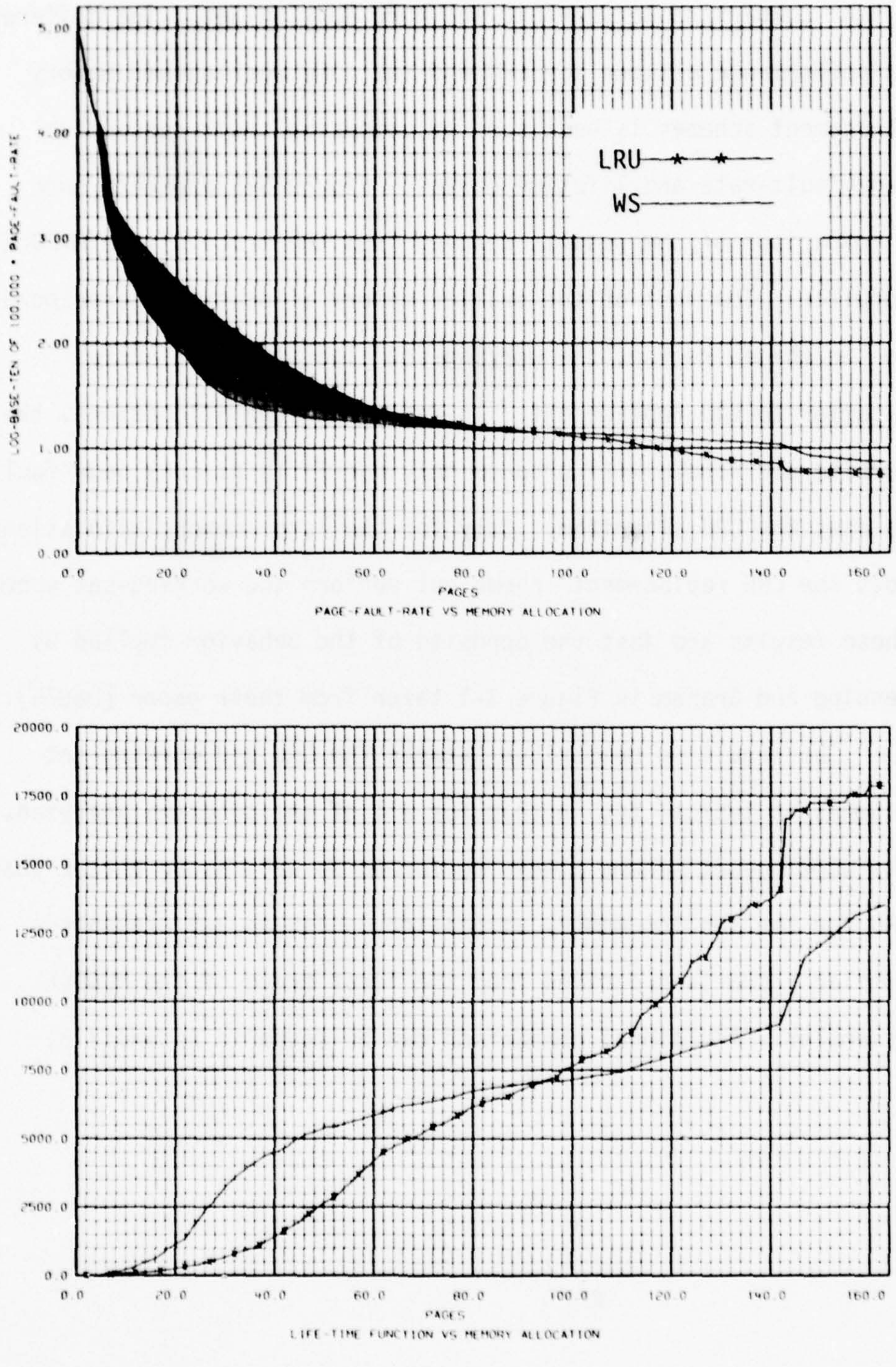


FIGURE 3-8 Comparison of LRU and Pure WS for MUDDLE Compiler

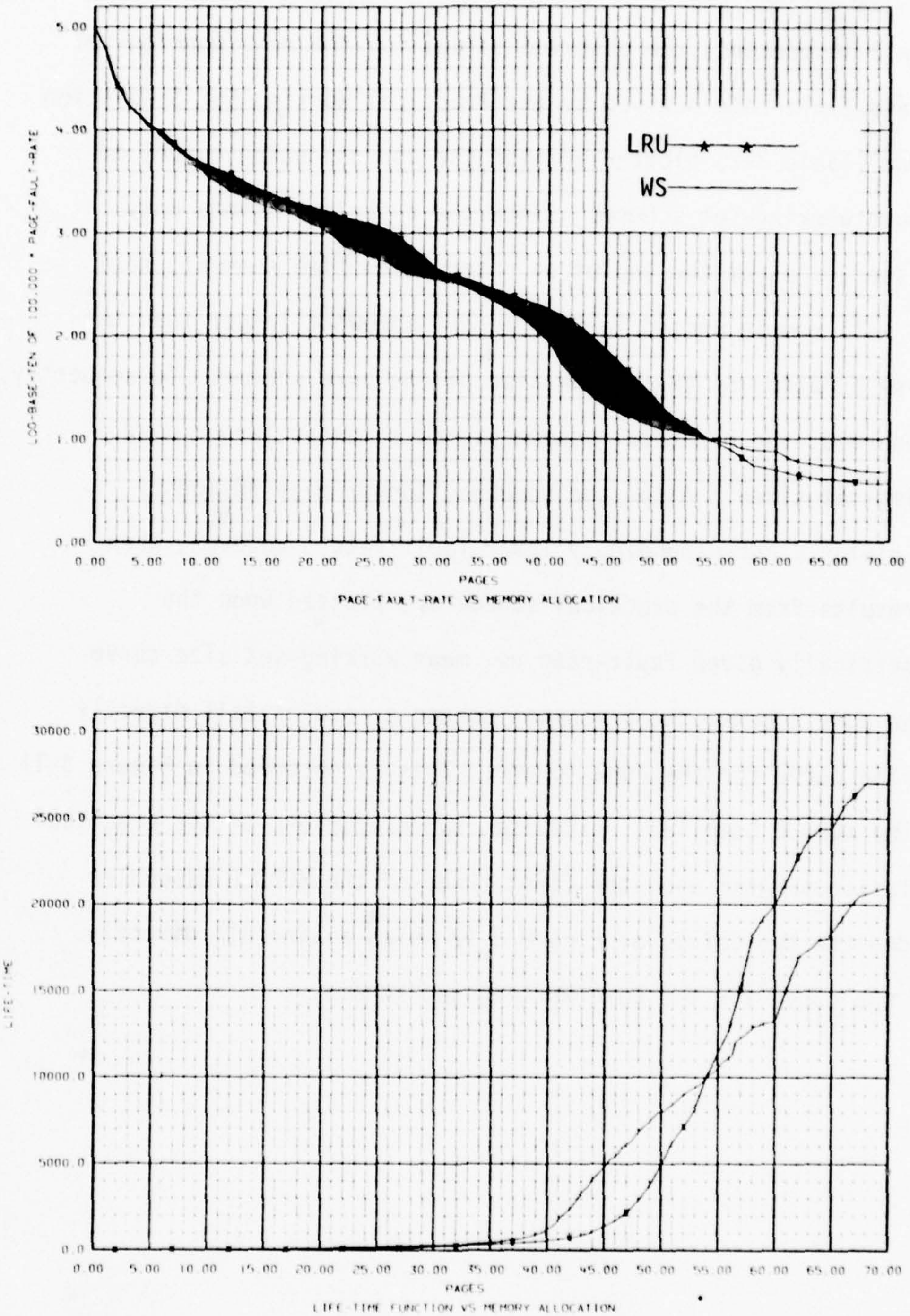


FIGURE 3-9 Comparison of LRU and Pure WS for MUDDLE Assembler (First Trace)

3.3.5 Comparison Between Practical and Pure Working-Set Schemes. Figure 3-10 presents the observed values for the mean working-set size and fault-rate derived by the Practical Working-Set Simulation Program (Table 3-2) plotted against the corresponding curves of the pure working-set scheme. Since the practical scheme only dismisses pages at the end of an observation time segment, some pages will remain unused in main memory somewhat longer than the precise duration of the window size of the pure scheme. Consequently, as observed, the practical scheme yields somewhat larger mean working-set sizes. Also, the somewhat longer page residency time yields a correspondingly lower fault-rate. However, when the results from the practical scheme are plotted upon the parametrically given fault-rate vs. mean working-set size curve of the pure scheme, the practical scheme's results fall directly upon the curve for the pure scheme. This is observed in Figure 3-11 for the MUDDLE Compiler. Likewise, two executions of the practical simulation program upon the first trace of the MUDDLE Assembler yielded the two points of Figure 3-12 which again fall directly upon the curve for the pure working-set scheme.

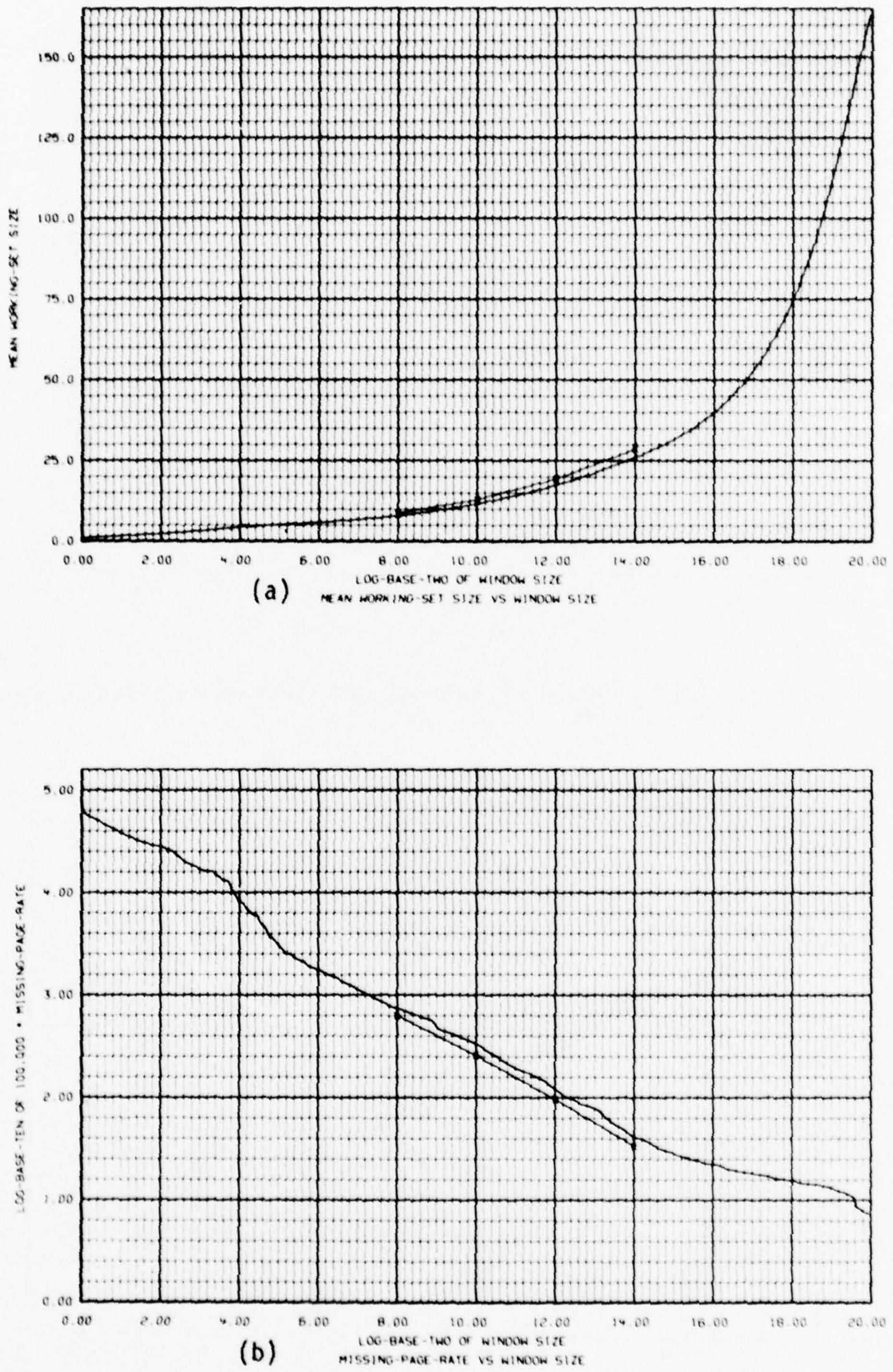


FIGURE 3-10 Preliminary Comparisons of Practical and Pure WS for MUDDLE Compiler

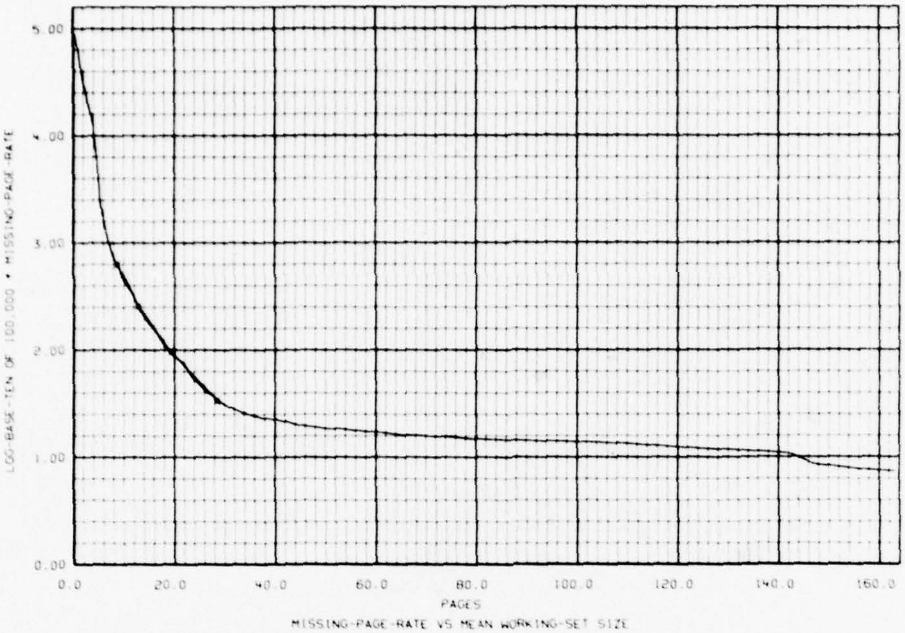


FIGURE 3-11 Comparison of Practical and Pure WS for MUDDLE Compiler

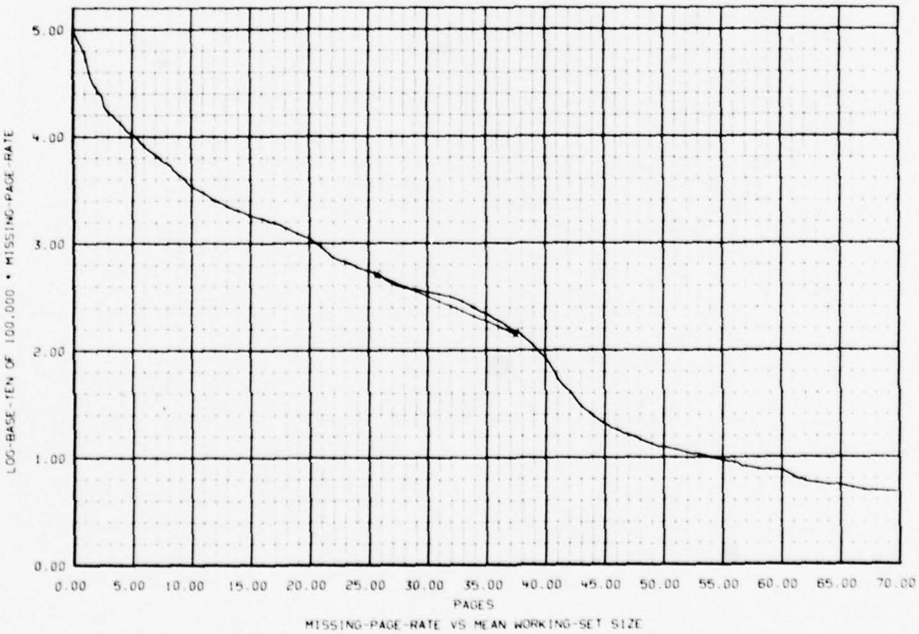


FIGURE 3-12 Comparison of Practical and Pure WS for MUDDLE Assembler (First Trace)

3.4 Conclusions

The major conclusions which can be drawn from the summary results of this Chapter and the substantially larger collection given in Appendix B are as follows:

1) For many programs, a pure working-set memory management scheme is as good as or better than an LRU scheme for smaller memory allocations. Moreover, the Denning-Graham view of comparative performance [DeG75] is just the opposite from that which has been observed.

2) Results from a practical implementation of a working-set policy fall directly upon the fault-rate vs. mean working-set size curve of the ideal (pure) scheme.

3) Many programs have very definite phases in their execution. Moreover, the working-set sizes are not Gaussian but are rather bimodal or trimodal.

4) The use of a logarithmic scale for plotting values of the page-fault-rate function is mandatory to preserve sufficient details. The use of a log-base-two scale for the window size is also very useful when plotting working-set statistics.

CHAPTER 4

SECOND ORDER STATISTICS OF PROGRAM BEHAVIOR

4.1 The Need for Second Order Statistics

The existance of numerous texts (e.g., [Che65], [LAG68], [WeR73]) on the use of computers for on-line process control attests to the fact that the subject area is a well developed discipline. Here, the term 'process' is used in the broadest sense of the word and may refer to an engineering system (e.g., chemical plant, aircraft, control system), to a biological system (e.g., human tracking action) or perhaps even to an economic or sociological system. For engineering systems, the computer has in fact become an intimate part of the overall system. Figure 4-1 taken from [Che65] provides a typical engineering description of the use of a computer for process

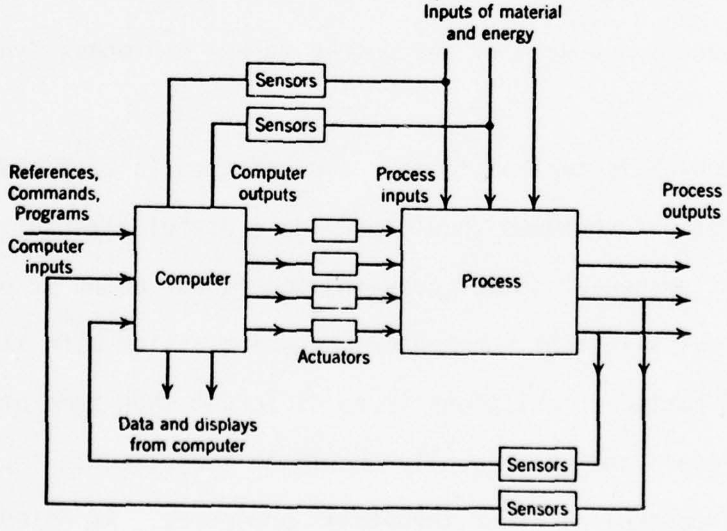


FIGURE 4-1 Description of Use of a Computer for Process Control

control. The process has as its primary inputs the materials and energy that are necessary to produce its outputs. Sensors on the process inputs and outputs are used to provide feedback signals which are computer processed into actuator commands. The process to be controlled is driven by a set of actuating signals to yield a set of process outputs which are to be related in some specified way to the systems inputs.

The task of the designer is to measure (in so far as possible) the inputs and responses (outputs) and then from these measurements, to generate in the computer the required actuating signals. However, as noted by a leading control theoretician [Tru61]: "Intelligent design of a feedback control system can be effected only if the designer is cognizant of the dynamic characteristics of the process (or system) to be controlled: indeed, to a considerable degree, the extent to which system design can proceed in a logical and intelligent manner is directly measured by the degree to which process dynamics are known."

A theme which is central to this dissertation is that modern control-theoretic techniques should provide a useful alternative to the current preponderance of queueing models for computer operating systems. We maintain that a computer operating system with its collection of hosted computations is no different than some other industrial process and consequently should be subjected to the same rigor of control as other industrial processes. As noted earlier in Chapter 2 (Section 2.2), other have suggested control-theoretic approaches or noted their absence in a systematic analysis of computer systems. However, as also noted earlier in Chapter 1, an immediate consequence of process control considerations is the recognition of the current lack of validated dynamical models for the behavior of operating systems from a resource utilization viewpoint. Earlier papers by Rodrequez-Rosell [R-R72] and Wilkes [Wil73] have also noted the need for the "equations of motion" and the "plant dynamics" of an operating system.

If a process control engineer was given the task of improving or designing anew a control scheme for some industrial process, his first concern, as noted above, would be to determine an appropriate set of state variables, control variables and the associated process dynamics which connect them. If such dynamical models did not exist or were deemed inadequate, his next order of business would be a measurement program upon the specific system if it existed, on similar systems, or possibly upon some pilot plant which would have to be first constructed. If the subject process happens to be a computer operating system, a process control engineer would not wring his hands as some computer scientist has done at "the futility of trying to write the system's equations" [R-R72] but would rather immediately embark upon some measurement program from which dynamical models could be validated. It is our contention here that the computer scientist too should get on with a measurement program guided by the same issues which motivates a process control engineer.

New measurements are required because past measurements upon operating systems have not been of the type required for validating the desired input-output system models. The lack of such measurements is seen as a direct consequence of the dominance of queueing models. Guided by queueing type models, past measurements have been primarily direct toward first-order statistics, such as means, variances, and distributions for job arrivals, job delays, job demands, etc. As we shall see below the application of modern estimation and control-theoretic techniques requires second-order statistics, such as the auto- and cross-correlations of the processes involved.

Returning to the important problem of system (process) identification, the specific measurements a process control engineer will make upon some "plant" are determined by the engineer's prior selection of some class of system models (i.e., linear) and upon his view of the measurement process itself. Although the general (parametric) form of the system model may be known or assumed, the presence of noise disturbances to the process or the effect of certain environments, which are difficult if not impossible to measure, may cause the specific parameters of the model to change with time. Furthermore, the measurement process itself may be subjected to additive noise. In any case, the presence of noise accentuates the process identification problem and reduces the degree of precision obtainable in establishing the accuracy of the parameters of a dynamical model of the process. As such, the control engineer is often prepared to smooth and filter the measurement data with the implicit understanding that his overall system performance goal can now only be met in some average (e.g., mean square) sense.

For such noisy or stochastic processes, the communication and control engineers have developed an extensive set of tools and estimation techniques (e.g., optimal Wiener or Kalman-Bucy filters) [Lee60] [BuJ68] [McG74]. At the heart of all these modern techniques are the second order (joint) probability distributions of the processes involved and their associated statistics such as the auto- and cross-correlations, the auto- and cross-covariances, and the auto- and cross-spectral density functions.

Consequently, if one hopes to employ the process control engineers's approach to computer operating system, one must select some appropriate set of state-variables and then make the proper set of measurements upon these variables such that their auto- and cross-correlations functions can be estimated. Having characterized the state-vector of an operating system in Chapter 1 as the set of utilization levels of each of the system's resources, a full set of measurements should ideally allow for the estimation of both the first and second order statistics of the individual resource utilization levels. Specifically, one would like the autocorrelation for each resource process as well as all their cross-correlations (e.g., the cross-correlation of memory to CPU utilization). Unfortunately, such a complete set of measurements upon an operating system would be a major undertaking and would represent several years work and considerable computer processing time in reducing the data.

The merits of a control-theoretic approach can, however, certainly be demonstrated upon a more limited set of measurements. Consequently, we have limited our current investigation to a single resource, the memory component, which is the most expensive and most studied of the system's resources. And yet, for all the studies of memory utilization behavior and memory management techniques, except in two instances, there have not been any measurements estimating second-order statistics. In [LeY71], Lewis and Yue considered the statistical characteristics of stack distance strings. Specifically, they estimated the cumulative periodogram (i.e., the cumulative sample auto-spectral density function) of the distance strings derived from three different programs. Their results reject an assumption of serial independence in the distance strings. In [Bry75], Bryant has estimated the autocorrelation funtions for the working-set size and its low order differences (i.e., Δw, $\Delta^2 w$, $\Delta^3 w$) for several programs. Other published empirical data on memory behavior is limited and except for the two cases noted, has been confined to first-order statistics — such as sample means, variances, and histograms — of working-set size, page reentry rate, missing page probability, etc. With a view toward determining better models for memory behavior and better memory management (control) algorithms, this chapter presents some empirical second-order statistics (autocovariance functions) which have been estimated from the same trace data as was used for the first order statistics presented in Chapter 3.

4.2 A Definition of Locality

Experimental studies of program behavior (e.g., [Lip68], [Hat72], [R-R73], [Oli74]) have repeatedly shown that for most practical programs, a program tends to favor a subset of its pages during any time interval and that the membership of the set of favored pages tends to change slowly. Attempts to formalize this empirical observed behavior has been known as the Principle of Locality. The original discoverer of this principle seems to have been lost but as Madnick observes [Mad73], "its definition has changed in time". In fact, an examination of all references on locality will show that very few have precisely the same definition (e.g., c.f. [Den58c], [Den70], [DeS72], [CoR72], (Den72], [DDS72], [SpD72], [ShT72], [Mad73]). Some definitions are directed toward basic program behavior while others address paged memory behavior.

As an initial model for program behavior, the definition of Spirn and Denning [SpD72] was selected as our working definition of locality. While the selected definition will be shown to be false, its selection was prompted by two considerations. First, it is phrased in terms of paged memory behavior which is closer to memory management considerations rather than just program behavior. Secondly, it is an operational definition in terms of a second order statistic, namely correlation. For completeness, we restate Spirn and Denning's definition below using a slightly different notation which will improve some of our later definitions.

Thus, consider an N-page program whose pages p_n constitute the set

$$P = \left\{ p_1, p_2, \dots p_N \right\}.$$

The reference string $r_1, r_2, \dots r_t \dots$ is a sequence generated from the members of P and such that the value of the reference r_t is the specific page p_k containing the address referenced at time t — time being measured in terms of the number of memory references made by the program. Suppose the reference string has been divided up into intervals, each of length T called the interval size. Let L(i) be the observed locality — the set of pages referenced — in the i-th interval. Then with respect to the given sequence of intervals and localities,

$$L(1), L(2), \dots, L(i), \dots$$

one has the following:

Definition (Spirn and Denning [SpD72]) A reference string is considered to satisfy the property of locality if for its associated locality sequence,

1) For almost all i, L(i) is a proper subset of P;

2) For almost all i, L(i) and L(i+1) tend to have many pages in common; and

3) The observed localities L(i) and L(i+j) tend to become uncorrelated as j becomes large.

For our purposes, we note that condition (2) could be rephrased as,

2') Adjacent localities such as L(i) and L(i+1) are highly correlated.

Spirn and Denning, however, fail to define the concept of correlation between localities. For a concept which is so central to this study we quickly remedy this situation below.

4.3 Autocorrelation Statistics of Paged Program Behavior

4.3.1 Autocorrelation of Localities. For a program P of N pages, the range space $\mathcal{L}$ of the mapping L(i) is the power set [Hal60] of P consisting of the 2^N (2^N-1 if we exclude the null locality) possible subsets of P, i.e.,

$$\mathcal{L} = 2^P = \left\{ L_0, L_1, L_2, \ldots, L_{2^N-1} \right\}. \tag{4.1}$$

Moreover, there is a natural isomorphism between the elements of $\mathcal{L}$ — possible values of L(i) — and a set of N-bit binary numbers $\underset{\sim}{b}_i$. The bit in position k being a one if page p_k was referenced during the observation interval or otherwise, zero.

As an example, for an eight page program (N=8) whose observed localities were

$$\begin{aligned}
L(1) &= \{p_1, p_4, p_6, p_7\} = L_{150} \\
L(2) &= \{p_1, p_2, p_4, p_5, p_7\} = L_{218} \\
L(3) &= \{p_2, p_5, p_8\} = L_{73} \\
\ldots & \qquad \ldots \\
L(i) &= \{p_1, p_3, p_4, p_6, p_8\} = L_{181}
\end{aligned}$$

one has the binary numbers

$$\underset{\sim}{b}_1 = 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 = 150_{10}$$

$$\underset{\sim}{b}_2 = 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0 = 218_{10}$$

$$\underset{\sim}{b}_3 = 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1 = 73_{10}$$

$$\cdots \qquad \cdots$$

$$\underset{\sim}{b}_i = 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 = 181_{10}$$

The concept of correlation is most often introduced in terms of continuous [Lee60] or discrete [Gol74] time waveforms rather than in terms of sets (localities). Upon observing that the binary numbers (vectors of zeros and ones) $\underset{\sim}{b}_i$ can be viewed as binary waveforms $b_i(n)$ in bit serial order, one has the three way isomorphism

$$L(i) \leftrightarrow \underset{\sim}{b}_i \leftrightarrow b_i(n) \qquad (4.2)$$

which ultimately relates localities to binary waveforms. Figure 4-2 presents the set of waveforms which correspond to the example above.

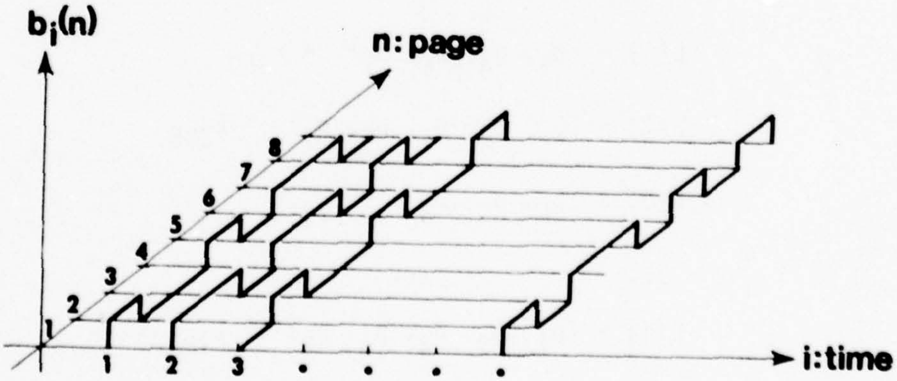


FIGURE 4-2 Time History of Binary-Valued Waveforms Corresponding to Program Localities

Having related waveforms to localities, one could thus define the correlation $\emptyset_{ij}$ between localities L(i) and L(j) as equivalent to the correlation between the binary waveforms of their respective isomorphic images $\underset{\sim}{b}_i$ and $\underset{\sim}{b}_j$, i.e.,

$$\emptyset_{ij} = \left\langle \underset{\sim}{b}_i, \underset{\sim}{b}_j \right\rangle = \sum_{n=1}^{N} b_i(n) b_j(n); \tag{4.3}$$

the value (coefficient) of correlation being the inner product of the associated binary valued vectors. This definition of correlation is nonstatistical and is equivalent to the concept of correlation between code words in coding theory. For purposes of future operating system design, one must expand the definition of correlation to give it its full statistical basis.

The design of future memory management schemes will not depend upon the specific correlation between any two localities, but rather upon their average correlation value. Moreover, operating system design cannot identify or benefit from a unique epoch in time. Therefore, one must assume time stationarity for the reference (or locality) strings and consequently, one's measure of correlation between localities should be a function of only time differences (relative time). Hence, as a first definition, consider

Definition I. The autocorrelation $\Phi(j)$ between observed localities separated by j intervals is given by an average of coefficients $\emptyset_{i,\, i+j}$ over all intervals i, i.e.,

$$\Phi(j) = \underset{i}{\text{Ave}} \left\{ \left\langle \underset{\sim}{b}_i, \underset{\sim}{b}_{i+j} \right\rangle \right\}. \tag{4.4}$$

4.3.2 Average Page Autocorrelation. The complete set (family) of binary waveforms $b_i(n)$ of Figure 4-2 can be considered as a sampled (in time) version of a two dimensional function $B(t, n)$ as given in Figure 4-3 which completely describes the paged memory behavior of a program over its execution history. An alternative to the locality view is provided by considering the function $B(t, n)$ to be sampled (in memory space) into individual page behavior histories, say

$$\beta_n(t) = B(t, n). \tag{4.5}$$

The resulting $\beta_n(t)$ waveforms could also have been introduced directly wherein the value of $\beta_n(t)$ is equal to one during those intervals in which the n-th page is referenced and zero otherwise. Figure 4-4 presents the time histories of the $\beta_n(t)$ waveforms corresponding to our earlier example.

The complete set of N temporal functions $\beta_1(t)$, $\beta_2(t)$, ..., $\beta_N(t)$ also constitute a possible set of state-variables for representing the memory subsystem behavior. By also sampling the β_n waveforms in time, we have the equality,

$$\beta_n(i) = \beta_n(i\Delta t) = B(i\Delta t, n) = b_i(n), \tag{4.6}$$

which demonstrate the equivalence of the two different state-variable representatives. In fact, the double (two-dimensional) sampling of the $B(t, n)$ function (surface) produces the matrix of sample values

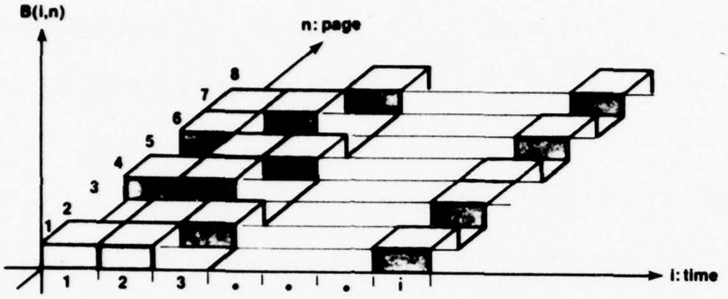


FIGURE 4-3 Time History Surface Corresponding to a Program's Memory Usage

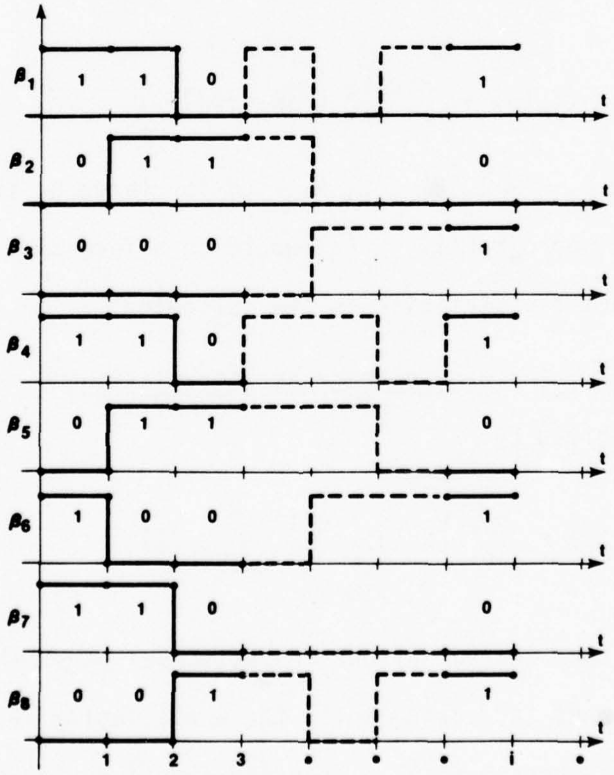


FIGURE 4-4 Time History Waveforms Corresponding to Individual Page Usage

presented in Figure 4-5 which also completely describes the paged memory behavior of a program over its executive history of M time intervals. The equivalence (4.6) is nothing more than the equivalence between a row (locality) or column (individual page) perspective of the same matrix.

Since B(t, n) (or either family of $b_i(n)$ or $\beta_n(i)$) is a function of both time t (or i) and memory space n, one could conceivably introduce both temporal and spacial correlations. However, for prediction purposes, one would like only temporal correlations. Thus quite naturally from this second viewpoint, one could also introduce the whole family of correlation functions

$$\Psi_n(j) = \underset{i}{\mathrm{Ave}} \left\{ \beta_n(i)\beta_n(i+j) \right\} , \qquad (4.7)$$

one for each page n, n = 1, 2, ..., N. Yet for large N, the estimation of the whole family $\Psi_n(j)$ would be a formidable computational task. A more tractable statistic is the following:

<u>Definition II</u>. The <u>average page autocorrelation function</u> is given by

$$\Psi(j) = \underset{n}{\mathrm{Ave}} \left\{ \Psi_n(j) \right\} = \frac{1}{N} \sum_{n=1}^{N} \Psi_n(j). \qquad (4.8)$$

One might assume that the variations in individual page behavior are not too different and consequently the whole family (4.7) should be replaced by its average (4.8). Then the function $\Psi(j)$ could be

$$
\begin{array}{cc|cccc|c|cc|}
 & & \beta_1(\ ) & \beta_2(\ ) & \beta_3(\ ) & & \beta_n(\ ) & & \beta_N(\ ) \\
 & & & & & & \updownarrow & & \\
\underset{\sim}{L}(1) \equiv \underset{\sim}{b}_1 & & B(1,1) & B(1,2) & B(1,3) & \cdots & B(1,n) & \cdots & B(1,N) \\
\underset{\sim}{L}(2) \equiv \underset{\sim}{b}_2 & & B(2,1) & B(2,2) & B(2,3) & \cdots & B(2,n) & \cdots & B(2,N) \\
\underset{\sim}{L}(3) \equiv \underset{\sim}{b}_3 & & B(3,1) & B(3,2) & B(3,3) & \cdots & B(3,n) & \cdots & B(3,N) \\
 & & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
\cline{3-9}
\underset{\sim}{L}(i) \equiv \underset{\sim}{b}_i & \leftrightarrow & B(i,1) & B(i,2) & B(i,3) & \cdots & B(i,n) & \cdots & B(i,N) \\
\cline{3-9}
 & & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
\underset{\sim}{L}(j) \equiv \underset{\sim}{b}_j & & B(j,1) & B(j,2) & B(j,3) & \cdots & B(j,n) & \cdots & B(j,N) \\
 & & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
\underset{\sim}{L}(M) \equiv \underset{\sim}{b}_M & & B(M,1) & B(M,2) & B(M,3) & \cdots & B(M,n) & \cdots & B(M,N)
\end{array}
$$

FIGURE 4-5 Matrix of Sampled Values from a Program's Paged Memory Usage History

viewed as representing the correlation behavior of a generic page of the program. Our results below will, however, demonstrate that individual page behaviors are quite different and that the concept of a generic page behavior is a poor one. While our real motivation for introducing the average page autocorrelation $\Psi(j)$ is given in Chapter 6, the next Section gives a partial justification.

4.3.3 Some Relationships. Using the equality (4.6) between $\beta_n(i)$ and $b_i(n)$, direct substitution in the above equation yields the following:

$$\Psi(j) = \frac{1}{N}\Phi(j) \tag{4.9}$$

In words, we have the resulting theorem,

THEOREM I. Except for a scaling factor, the autocorrelation of program localities is equal to the average page autocorrelation.

For comparison purposes, since (c.f., [Lee60], p. 56) for any correlation function,

$$|\phi(j)| \leq \phi(0), \tag{4.10}$$

one usually works with the normalized autocorrelation

$$\hat{\phi}(j) = \frac{\phi(j)}{\phi(0)} . \tag{4.11}$$

Consequently, from Theorem I, we have the

<u>COROLLARY</u>. The normalized autocorrelation of program localities is equal to the normalized average page autocorrelation, i.e.,

$$\hat{\Phi}(j) = \hat{\Psi}(j) . \tag{4.12}$$

Finally, we observe that both $\hat{\Phi}(j)$ and $\hat{\Psi}(j)$ represent a normalized double averages (i.e., over both memory space and time). Their only true distinction being the order in which the averages are performed and the associated viewpoints they provide.

4.4 Autocovariance Statistics of Paged Program Behavior

From a prediction and control viewpoint, neither of the statistical measures $\Phi(j)$ nor $\Psi(j)$ is ideal. The problem is that both the spacial locality vectors (b-vectors) and the temporal β waveforms have a mean value which biases the asymptotic values of their individual associated correlation functions and consequently also their second averages realized in $\Phi(j)$ and $\Psi(j)$. Our empirical results below show that the resulting asymptotic values of the correlation functions can be quite large (≈80% of maximum value) and thus hide much of the second order statistical behavior. The usual solution in such cases is to remove the mean value and then work with the autocovariance function rather than with the autocorrelation function. Consequently, we introduce the following definitions:

4.4.1 Average Page Autocovariance. For each page n of an N page program, define the individual page autocovariance functions as

$$\psi_n(j) = \underset{i}{\mathrm{Ave}} \left\{ \left[\beta_n(i) - \bar{\beta}_n\right]\left[\beta_n(i+j) - \bar{\beta}_n\right] \right\}. \qquad (4.13)$$

or in the finite sample case as

$$\psi_n(j) = \frac{1}{M} \sum_{i=1}^{M-j} \left[\beta_n(i) - \bar{\beta}_n\right]\left[\beta_n(i+j) - \bar{\beta}_n\right] \qquad (4.14)$$

Finally, as before, we introduce the more tractable statistic as follows:

Definition III. The *average page autocovariance function* is given by

$$\psi(j) = \underset{n}{\mathrm{Ave}} \left\{ \psi_n(j) \right\} = \frac{1}{N} \sum_{n=1}^{N} \psi_n(j). \qquad (4.15)$$

4.4.2 Autocovariance of Program Localities. Having defined the covariance statistic (4.15) related to our second correlation statistic (4.8), one might rightly ask what should be the corresponding covariance statistic when considering program behavior from our first (locality) viewpoint. One certainly would like to retain the reciprocity provided by the theorem of Section 4.3.3. Toward this goal, let us introduce the following:

Let the vector $\underset{\sim}{L}(i)$ correspond to the i-th row vector of the matrix of Figure 4-5, i.e.,

$$\underset{\sim}{L}(i) = \big(B(i,\ 1),\ B(i,\ 2),\ B(i,\ 3),\ \ldots,\ B(i,\ n),\ \ldots B(i,\ N)\big). \quad (4.16)$$

We shall call $\underset{\sim}{L}(i)$ the i-th locality vector which is completely equivalent to our earlier binary vector $\underset{\sim}{b}_i$ which was the isomorphic image of the i-th locality set $L(i)$. We also require the concept of a mean locality vector $\underset{\sim}{[}$ defined as

$$\underset{\sim}{[} = (\bar{\beta}_1,\ \bar{\beta}_2,\ \bar{\beta}_3,\ \ldots,\ \bar{\beta}_n,\ \ldots,\ \bar{\beta}_N) \quad (4.17)$$

where the $\bar{\beta}_n$ are the individual (column-wise) page usage means given by,

$$\bar{\beta}_n = \frac{1}{N} \sum_{n=1}^{N} \beta_n(i). \quad (4.18)$$

Finally, we introduce,

Definition IV. The autocovariance $\phi(j)$ between observed localities separated by j observation intervals is given by the following average,

$$\phi(j) = \underset{i}{\text{Ave}} \left\{ \left\langle \underset{\sim}{L}(i) - \underset{\sim}{[},\ \ L(i+j) - \underset{\sim}{[} \right\rangle \right\}, \quad (4.19)$$

or more explicitly in the finite sample case as,

$$\phi(j) = \frac{1}{M} \sum_{i=1}^{M-j} \left\langle \underset{\sim}{L}(i) - \underset{\sim}{[},\ L(i+j) - \underset{\sim}{[} \right\rangle. \quad (4.20)$$

4.4.3 Reciprocity Result. Corresponding to the theorem of Section 4.3.3, we have the following:

> THEOREM II. Except for a scaling factor, the autocovariance of program localities is equal to the average page autocovariance, i.e.,

$$\psi(j) = \frac{1}{N}\,\phi(j). \tag{4.21}$$

A proof is provided from the above definitions by the following development:

$$
\begin{aligned}
N\psi(j) &= \sum_{n=1}^{N} \psi_n(j) \\
&= \sum_{n=1}^{N} \frac{1}{M} \sum_{i=1}^{M-j} \left[\beta_n(i)-\bar{\beta}_n\right]\left[\beta_n(i+j)-\bar{\beta}_n\right] \\
&= \frac{1}{M} \sum_{i=1}^{M-j} \sum_{n=1}^{N} \{\beta_n(i)\beta_n(i+j)-\bar{\beta}_n\beta_n(i)-\bar{\beta}_n\beta_n(i+j)+\bar{\beta}_n\bar{\beta}_n\} \\
&= \frac{1}{M} \sum_{i=1}^{M-j} \sum_{n=1}^{N} \{B(i,n)B(i+j,\ n)-\bar{\beta}_nB(i,\ n)-\bar{\beta}_nB(i+j,\ n)+\bar{\beta}_n\bar{\beta}_n\} \qquad (4.22) \\
&= \frac{1}{M} \sum_{i=1}^{M-j} \left\{\left\langle \underset{\sim}{L}(i),\ \underset{\sim}{L}(i+j)\right\rangle - \left\langle \bar{\underset{\sim}{L}},\ \underset{\sim}{L}(i)\right\rangle - \left\langle \bar{\underset{\sim}{L}},\ \underset{\sim}{L}(i+j)\right\rangle + \left\langle \bar{\underset{\sim}{L}},\ \bar{\underset{\sim}{L}}\right\rangle\right\} \\
&= \frac{1}{M} \sum_{i=1}^{M-j} \left\langle \underset{\sim}{L}(i) - \bar{\underset{\sim}{L}},\ \underset{\sim}{L}(i+j) - \bar{\underset{\sim}{L}}\right\rangle \\
&= \phi(j)
\end{aligned}
$$

As before, we have the immediate corollary,

COROLLARY. The normalized autocovariance of program localities is equal to the normalized average page autocovariance, i.e.

$$\hat{\phi}(j) = \hat{\psi}(j) \tag{4.23}$$

4.4.4 Discussion. While the definition (Def. IV) for the autocovariance of program localities — once arrived at — seems quite natural, it must be admitted that our approach to it was less than straight forward. When our initial estimates for the correlation statistics indicated sizeable asymptotic values, it was clear that some mean value should be removed. Motivated by the computational consideration of not wanting to make two passes over the address trace-data, our first choice was to remove the means $\bar{b}_i$ of the individual locality vectors $\underset{\sim}{b}_i$, where

$$\bar{b}_i = \frac{1}{N} \sum_{n=1}^{N} b_i(n). \tag{4.24}$$

The removal of the running spacial mean yielded a normalized "pseudo-autocovariance" of program localities

$$\hat{C}(j) = \underset{i}{\text{Ave}} \left(\left\langle \frac{\underset{\sim}{b}_i - \bar{\underset{\sim}{b}}_i}{\underset{\sim}{\sigma}_i} , \frac{\underset{\sim}{b}_{i+j} - \underset{\sim}{b}_{i+j}}{\underset{\sim}{\sigma}_{i+j}} \right\rangle \right| \tag{4.25}$$

where $\underset{\sim}{b}_i$ and $\underset{\sim}{\sigma}_i$ are vectors of uniform elements $\bar{b}_i$ and σ_i, respectively; and $\bar{b}_i$ is the mean (4.24) and σ_i is the standard-deviation of the binary (locality) vector b_i. The statistic (4.25) was the basis of an earlier paper [ArG76] on program behavior. While the removal of the running spacial mean $\bar{b}_i$ did lower the resulting asymptotic values of the "pseudo-covariance", they were still substantial for some of the sample (address reference) data strings. We give only one sample result in Section 4.6.

Our initial disappointment with the removal of the individual spacial means $\bar{b}_i$ next prompted us to remove a total mean

$$\bar{B} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{n=1}^{N} B(i, n) \tag{4.26}$$

from the two dimensional waveform B(i, n). Estimates of the corresponding "covaraince" functions yielded effectively the same results as our first estimator (4.25) — no sample results are given.

Finally, the removal of neither the running spacial means $\bar{b}_i$ nor the total mean $\bar{B}$ could be justified when we choose to use our estimated statistics for prediction purposes. Only after a proper formulation of the memory usage prediction problem (see Chapter 6) was it clear that the covariance function $\phi(j)$ of Definition IV was the required statistic in spite of its computational demands of requiring two passes over the memory reference (data) strings — once first to estimate the mean locality $\bar{\underset{\sim}{L}}$ and then a second pass to estimate $\phi(j)$.

Our initial reluctance to introduce the concept of a mean locality was unfortunate. However, once having been recognized as a required concept for prediction purposes, it has subsequently been found to be another significant statistic of program behavior. In Section 4.6 below, results are presented on the variability of the mean locality as a function of window size.

4.5 The Estimation Programs

To investigate the correlation/covariance behavior of program localities, several additional analysis programs have been written which impelement the estimation of several of the above statistics. Generally, all the so-called Estimation Programs were written in FORTRAN and executed upon NUSC's UNIVAC 1108 computer. For input data, each program was designed to use the same trace tapes as described in Chapter 3 (especially Section 3.2.2). The analysis of any given program trace is controlled by three principal paramters, namely:

PGSIZE	- The page size
ISIZE	- The interval (window) size
LMAX	- The maximum lag index

Two sets of memory address bounds were also provided as input to allow each of the Estimation Programs to check that all addresses on the trace tape properly fell within the instruction or data segments of the traced program.

In analyzing a trace tape, each Estimation Program would build a list of page numbers referenced during each interval of ISIZE addresses. The page numbers were determined by dividing the address spaces of both the instruction and the data segments into pages of size PGSIZE. Executive function calls were counted but ignored. System I/O calls produced a termination of the current interval; the list of pages accumulated during the interval were excluded whenever the interval index at termination was less than 50 percent of the specified interval size.

At the termination of each observation interval, each program generated a vector of binary values (i.e., $\underset{\sim}{b}_i \equiv \underset{\sim}{L}(i)$) reflecting the specific page usage of the interval. Then depending upon the specific estimation program, various counts, sums, or lagged-products were accumulated from the individual locality vectors.

Finally, at the termination of the trace, each Estimation Program converted its accumulated counts, sums, products, etc. into the desired statistic. These results were tabulated along with other information such as number of intervals, number of executive function calls, number of I/O calls, etc. The resulting statistics were also plotted on an FR-80 plotting system. Each run (execution) of an estimation program produced but one plot of one statistic in a format essentially the same as that used in the multiple plot figures of the next section.

The specific estimation programs for which we show results in the next section were the following:

1) The Locality Autocorrelation Estimation Program which implemented the Definition I, (c.f. equation (4.4))

2) A Pseudo-Autocovariance Estimation Program which estimated the statistic of equation (4.25)

3) The Mean Locality Estimation Program which estimated the mean locality vector $\underset{\sim}{\bar{L}}$ as defined by equation (4.17).

4) The Locality Autocovariance Estimation Program which implemented the Definition IV, i.e., equation (4.20).

4.6 Presentation of Results

The Estimation Programs have been repeatedly applied to each of the seven data trace tapes for various combinations of the three principal control parameters. Variations of the page size (PGSIZE) by 50% or 200% had little effect on the resulting autocorrelation/autocovariance functions (only one result is given). As in Chapter 3, we have used smaller page sizes in the analysis of two of the smaller (traced) data programs than actually existed on the DMS PDP-10 system. *Consequently, the gross* statistics as presented earlier in Table 3-1 reamain the same.

Variations of the interval (window) size (ISIZE) produced major effects and thus is the controlling parameter in the parametric set of results which are presented. In presenting our results, we have consistently used the following lettering scheme to annotate the curves for different values of window (interval) size:

LEGEND	INTERVAL SIZE
A	64
B	256
C	1024
D	4096
E	16384
X	Intermediate Value

The choice of maximum lag index (LMAX) of 40 was selected with due consideration of the Estimation Program's running-time (over one hour for smaller intervals) and the four-for-one expansion of the interval size. In terms of relative time displacement, a lag of 40 for some interval size will correspond to a lag of 10 for the next larger interval size.

Figure 4-6 gives the normalized autocorrelation functions of program localities as estimated from the trace of the MUDDLE Assembler for observation interval (window) sizes 64, 256, 1024, 4096, and 16384. The results are, however, opposite to those of the expected behavior as implied by the definition of locality. As the interval size increases, the relative time difference between the observation intervals also increases and one would expect the degree of correlation to decrease. In fact, just the opposite effect is observed in that as the observation interval size is increased, the autocorrelation function yield larger (higher) asymptotic values. These larger asymptotic values must be attributed to the fact that for the longer observation intervals, the executing program consistantly requires a larger subset of its pages (i.e., a larger mean locality). The failure to remove the mean localities produces the sizeable asymptotic values approaching nearly 80 percent of maximum value.

Figure 4-7, taken from an earlier paper [ArG76] presents results comparable to Figure 4-6 but for a pseudo-autocovariance function (4.25) resulting from the removal of a running spacial (over memory) mean $\bar{b}_i$. While the asymptotic values are lower than those of Figure 4-6 they are still substantial. For some of the other sample traces (e.g., the MUDDLE Compiler) the pseudo-covariance

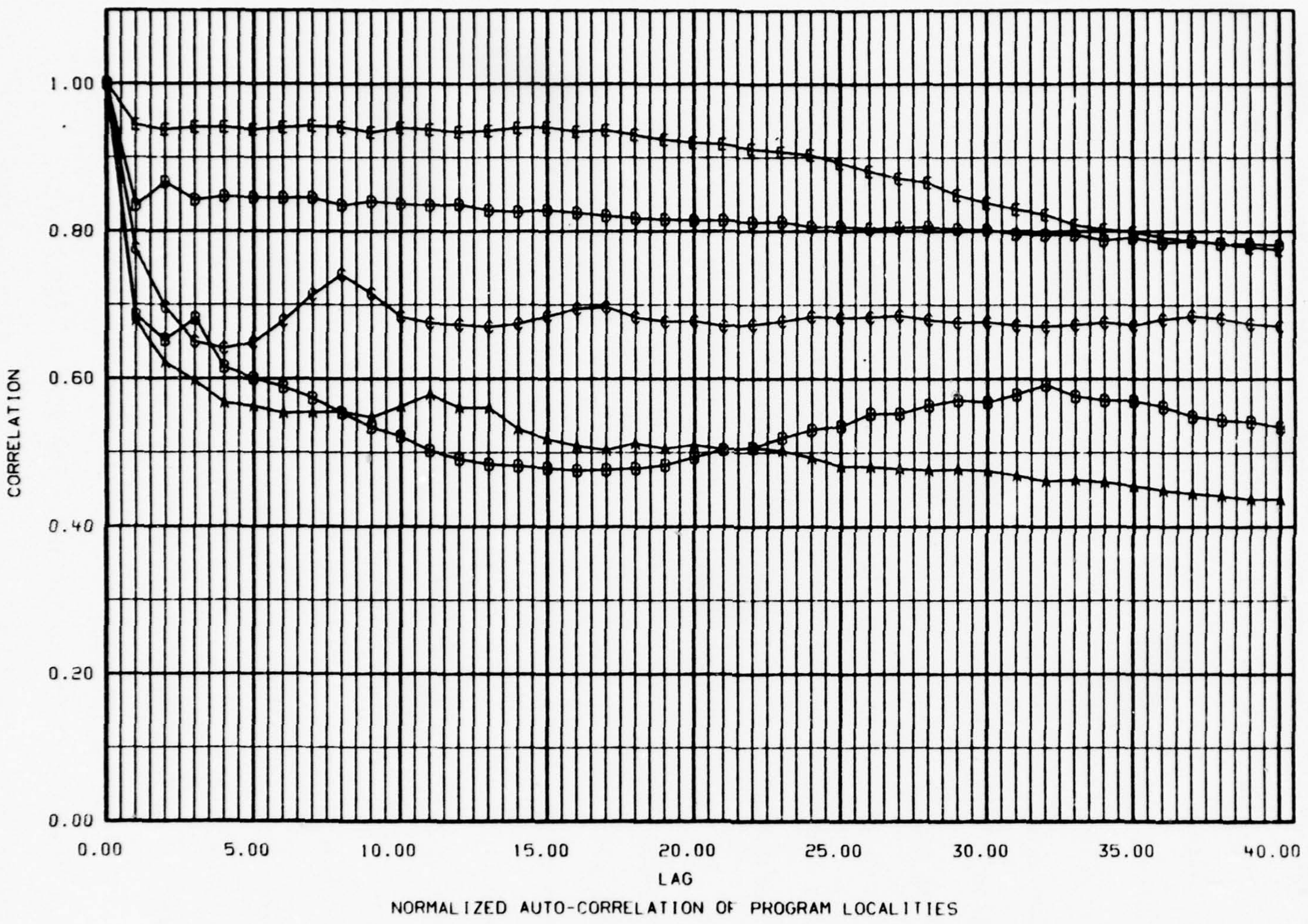


FIGURE 4-6 Autocorrelation of Program Localities for the MUDDLE Assembler (First Trace)

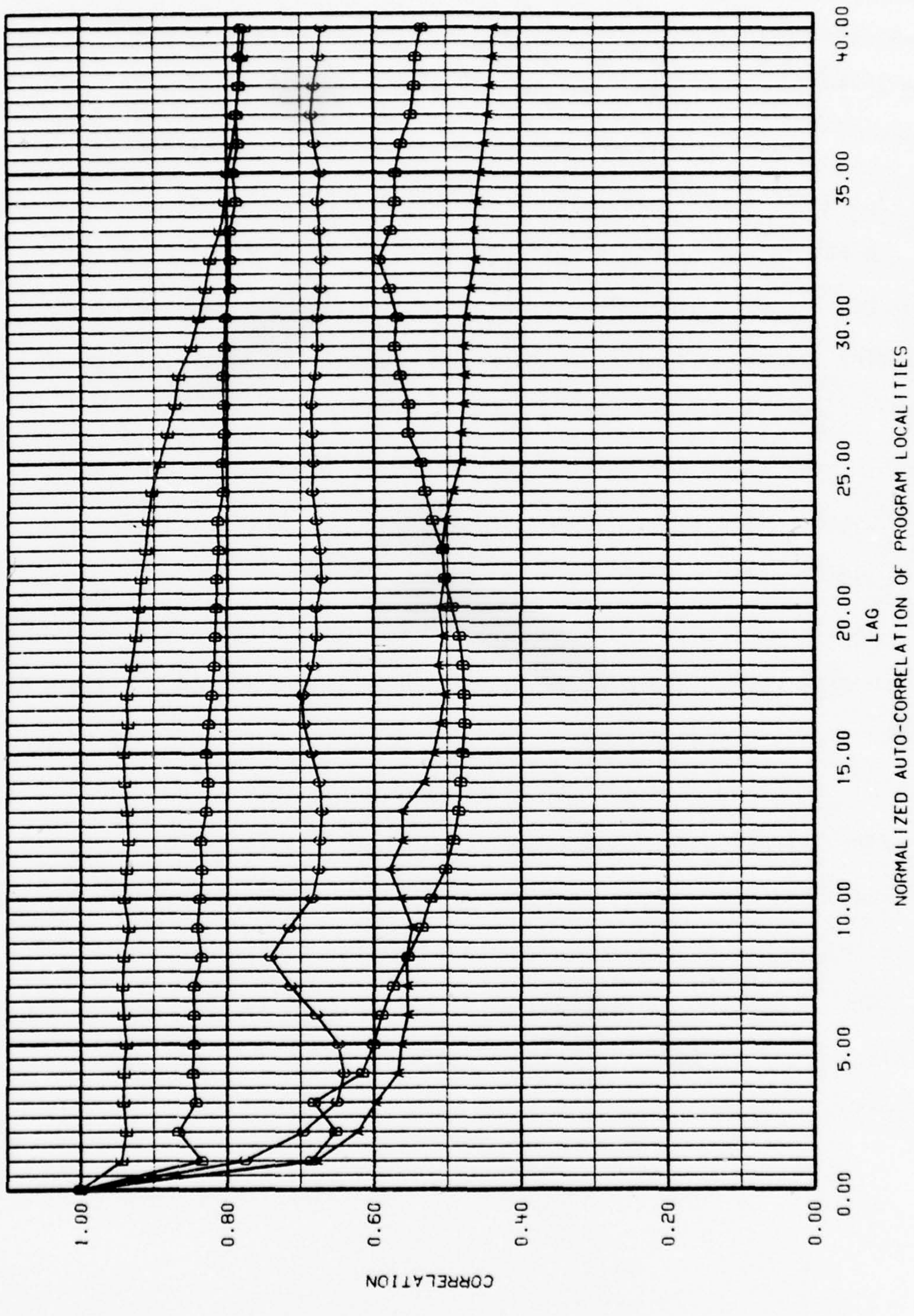


FIGURE 4-7 Pseudo-Autocovariance of Program Localities for the MUDDLE Assembler (First Trace)

statistic did produce quite usable results with small asymptotic values and which properly showed the structural detail of the true covariance function. As such, the pseudo-covariance statistic (4.25) is still a possible consideration in that computationally it saves a second pass over the address-trace data string.

To estimate the true covariance function of Definition IV, the Mean Locality Estimation Program was first repeatably executed to determine the mean locality vectors for each of the specified window sizes. Figure 4-8 presents the mean localities of the MUDDLE Assembler for three different window sizes as follows:

Case A:	Size = 64	Color = Blue
Case C:	Size = 1024	Color = Yellow + Blue
Case E:	Size = 16384	Color = Red + Yellow + Blue

One immediate observation is that some pages show a markedly different frequency of use depending upon the window size. Moreover, the prospect of realizing some generic page behavior is untenable.

Finally, using the estimated mean localities, the Autocovariance Estimation Program was repeatably executed to produce the set of curves of Figure 4-9. The resulting covariances for the MUDDLE Assembler of Figure 4-9 can be compared to the correlation and pseudo-covariance curve of Figures 4-6 and 4-7, respectively.

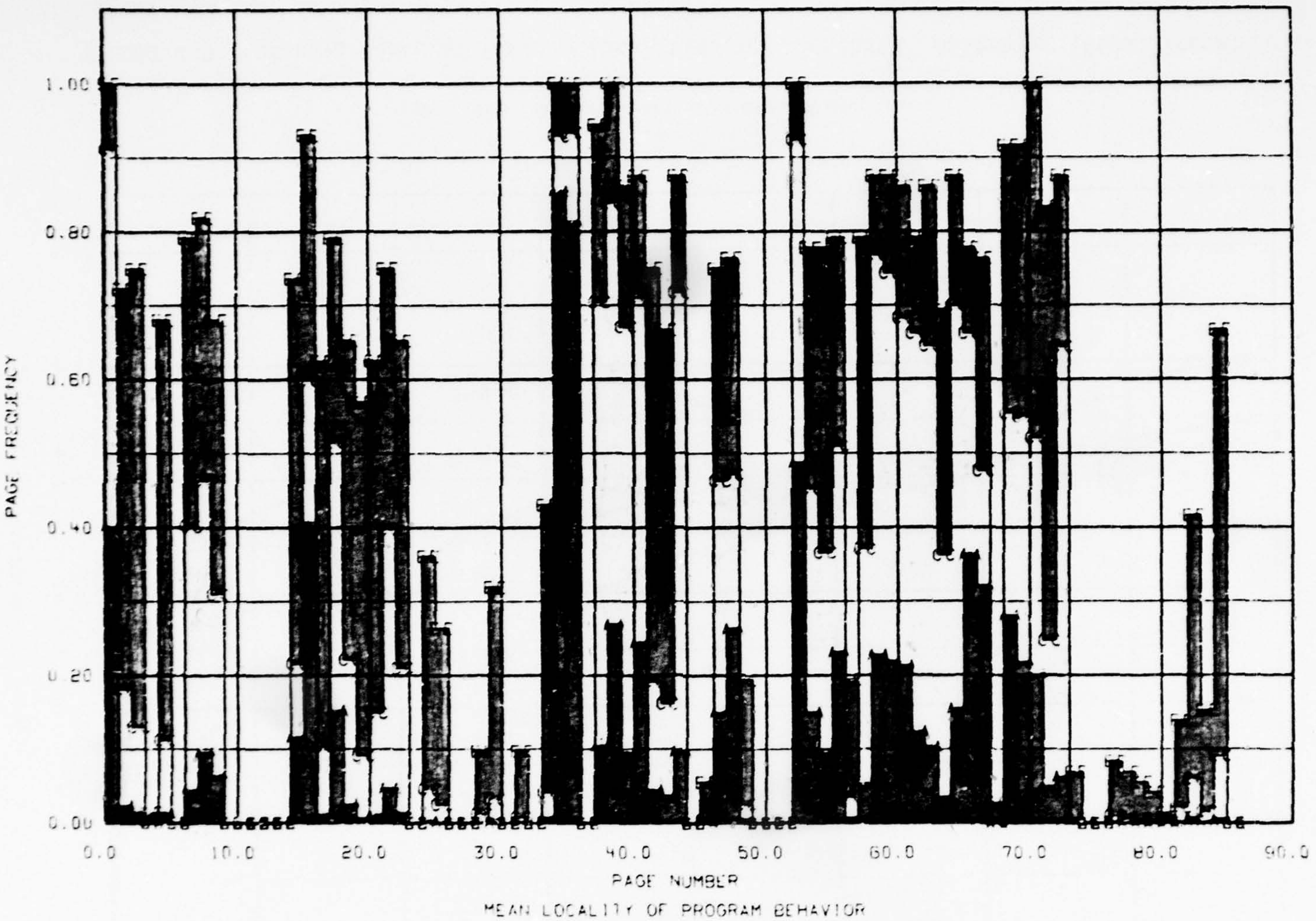


FIGURE 4-8 Mean Localities for the MUDDLE Assembler (First Trace)

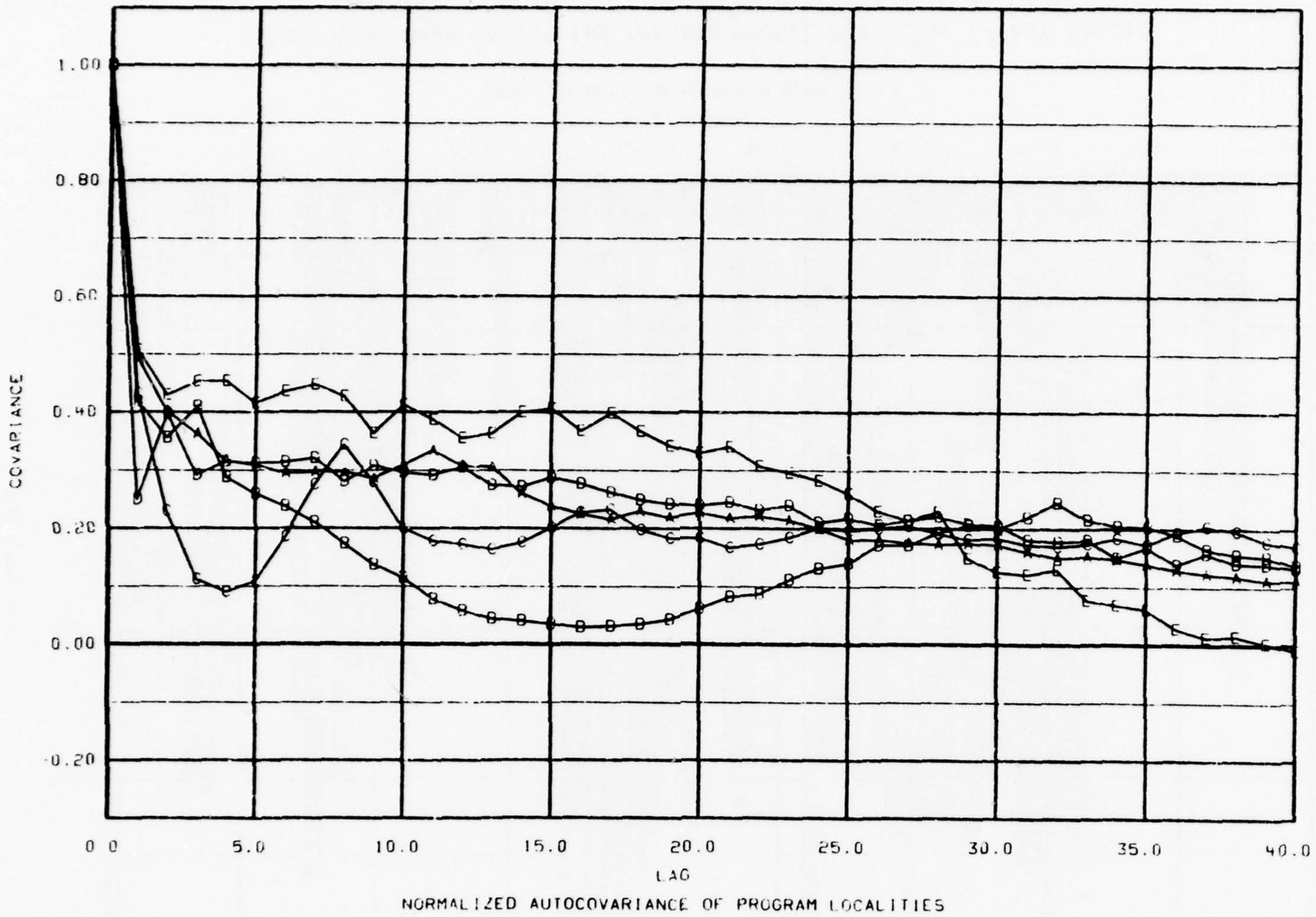


FIGURE 4-9 Autocovariance of Program Localities for the MUDDLE Assembler (First Trace)

The analysis of the second larger trace of the MUDDLE Assembler produced the results of Figures 4-10 and 4-11 which can be compared to the corresponding results (Figures 4-8 and 4-9, respectively) for the first trace. The mean localities do show some similarity between the two traces when allowance is made the difference in the number of pages (86 pages vs. 138 pages). The autocovariance functions however, show a marked similarity between the two traces and thus suggest that they are truly indicative of the correlation behavior of the MUDDLE Assembler.

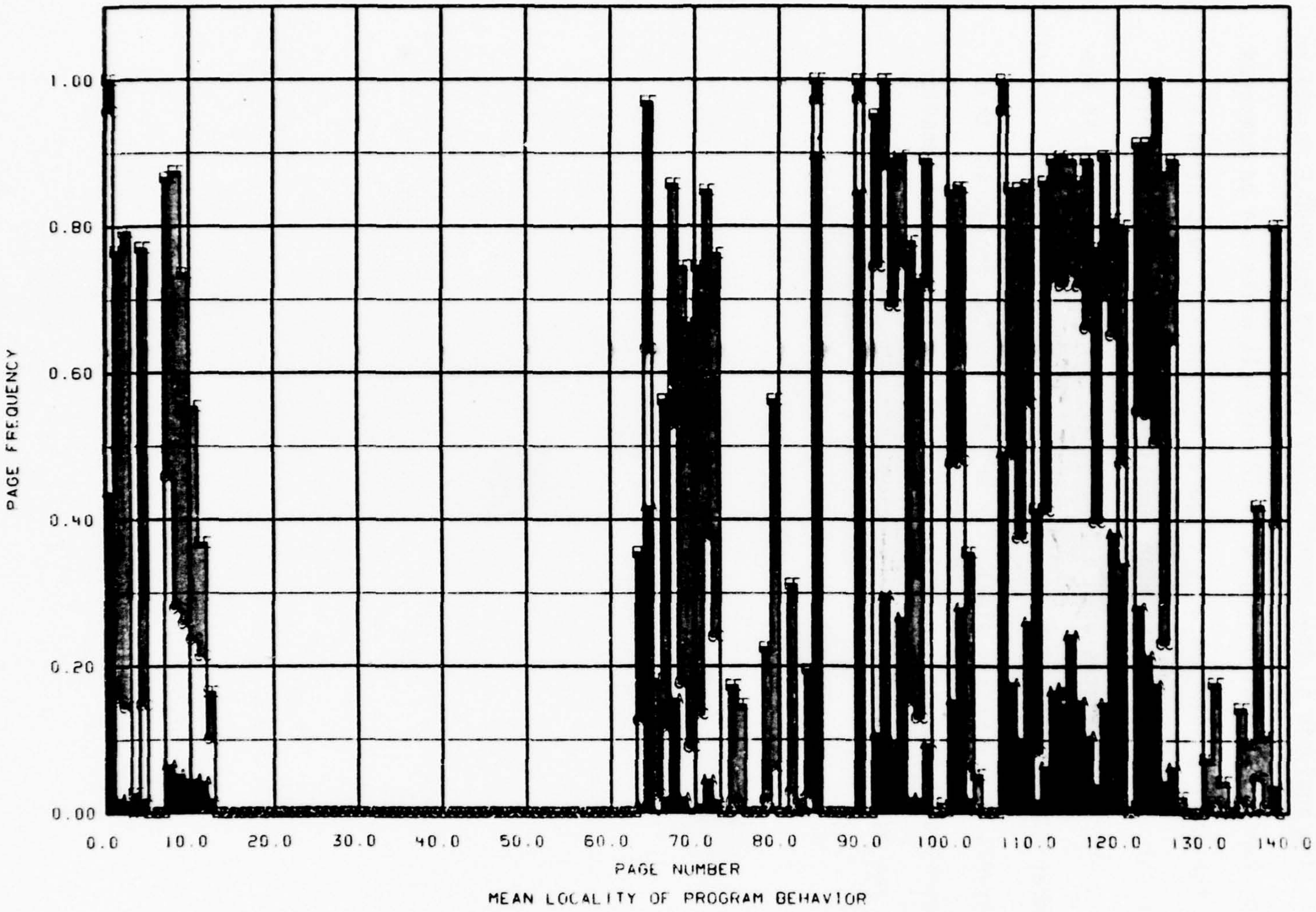


FIGURE 4-10 Mean Localities for the MUDDLE Assembler (Second Trace)

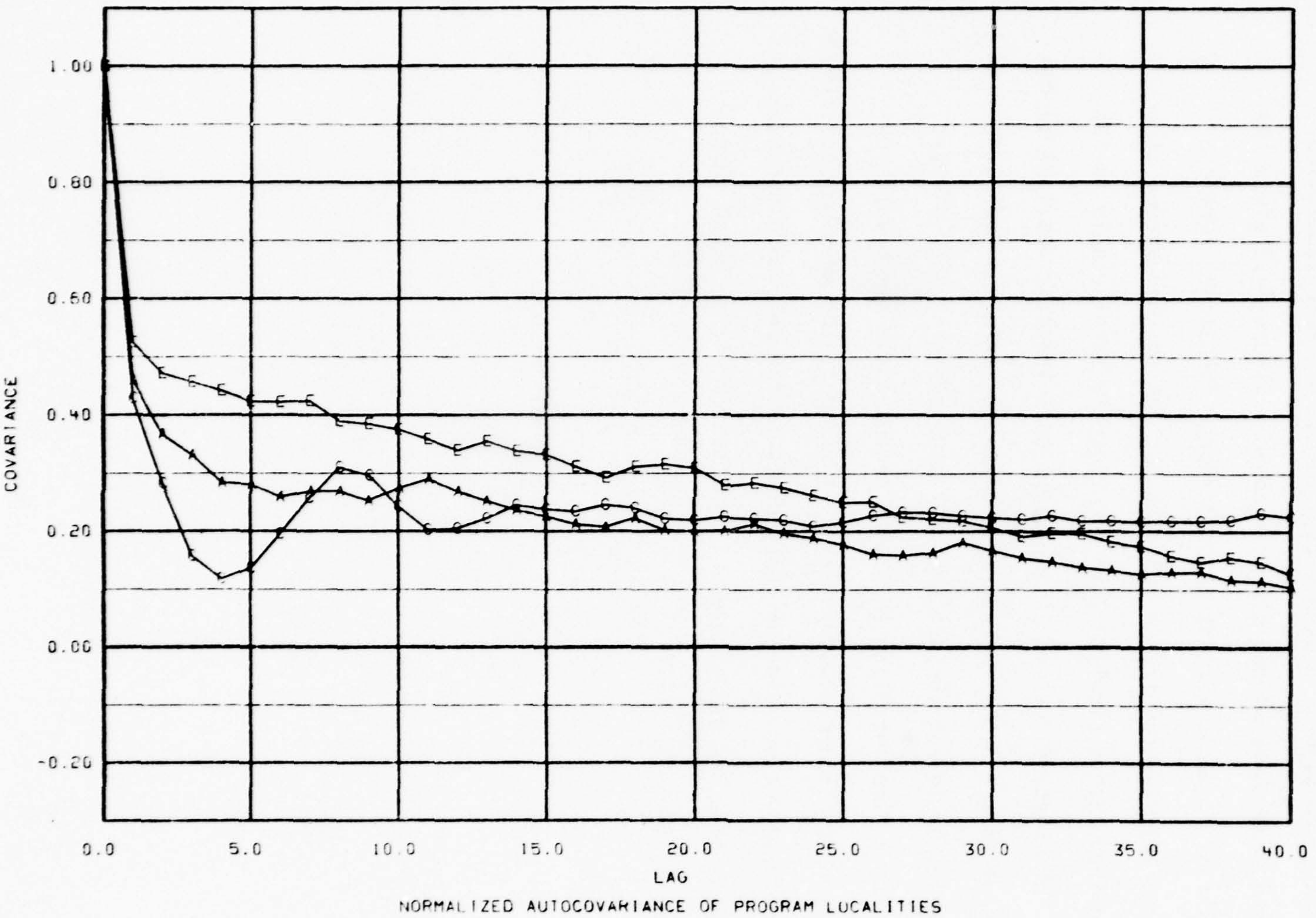


FIGURE 4-11 Autocovariance of Program Localities for the MUDDLE Assembler (Second Trace)

The next set of figures give results derived from the trace of the MUDDLE compiler. Figure 4-12 presents the mean localities for the same three window sizes as above. Figure 4-13 gives the resulting estimates for the autocovariance functions. One can observe that the covariance function for the smaller window size go to zero rather slowly. This is primarily attributable to the long term phase structure of the Compiler's trace as realized in the plots of Figure 3-6. The address reference string of 2,895,872 addresses would not support another quadrupling of the interval size but a final doubling to 32,768 produced the autocovariance function (of legend X) of Figure 4-14. Two of the previous covariance estimates are also repeated in Figure 4-14 for reference purposes. The peak at a lag of 36 intervals must reflect the second phase (pass) of the compiler.

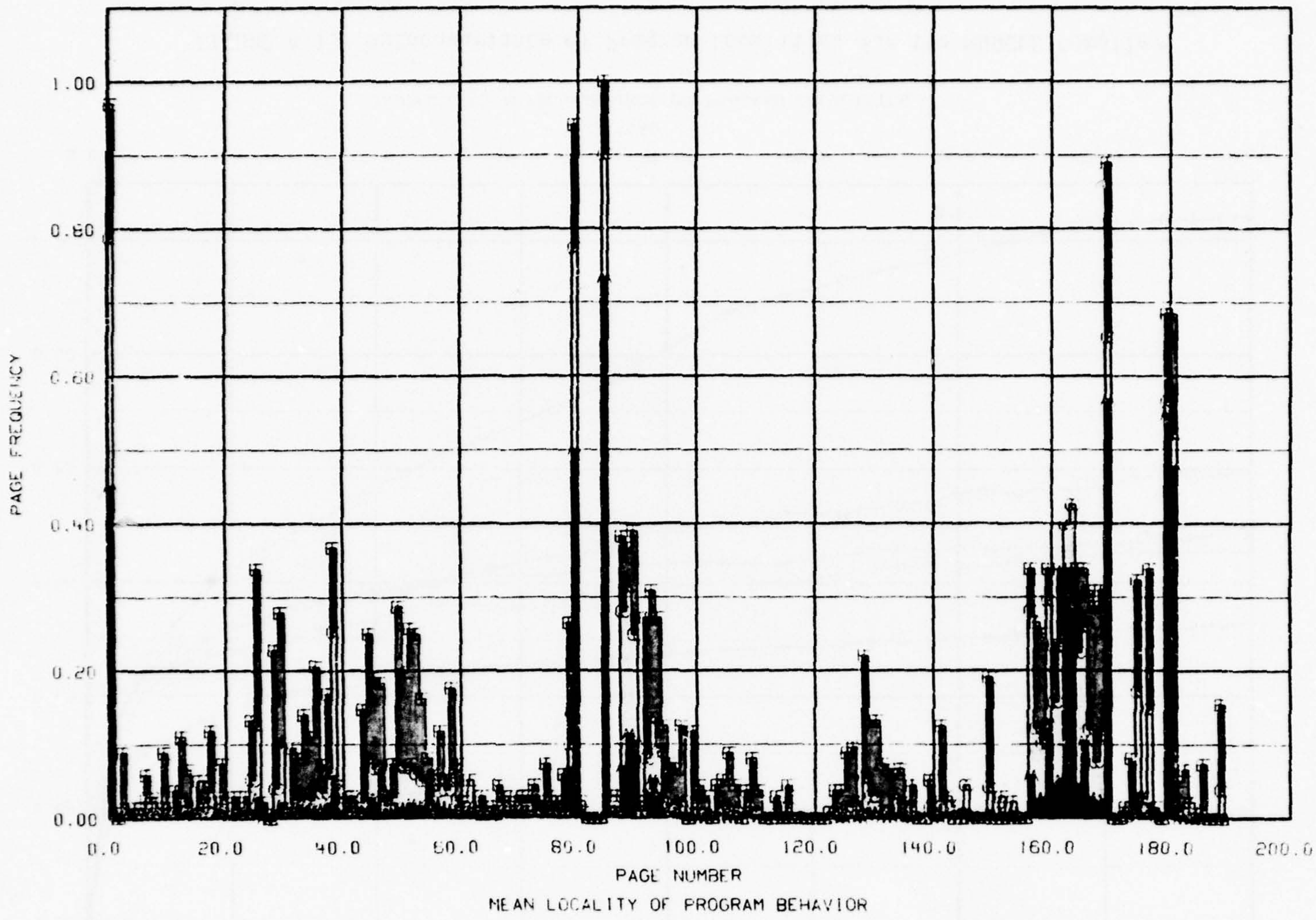


FIGURE 4-12 Mean Localities for the MUDDLE Compiler

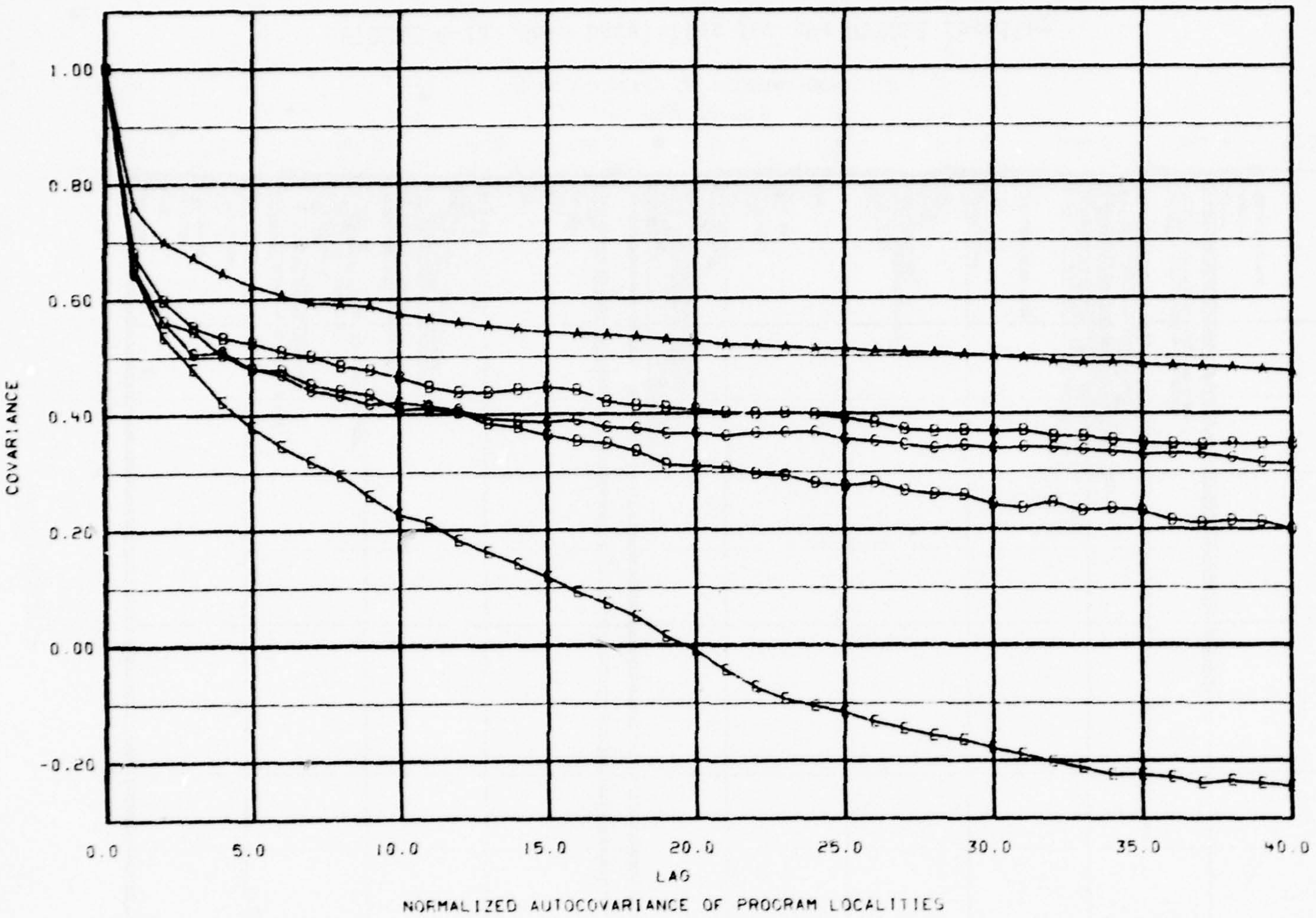


FIGURE 4-13 Autocovariance of Program Localities for the MUDDLE Compiler

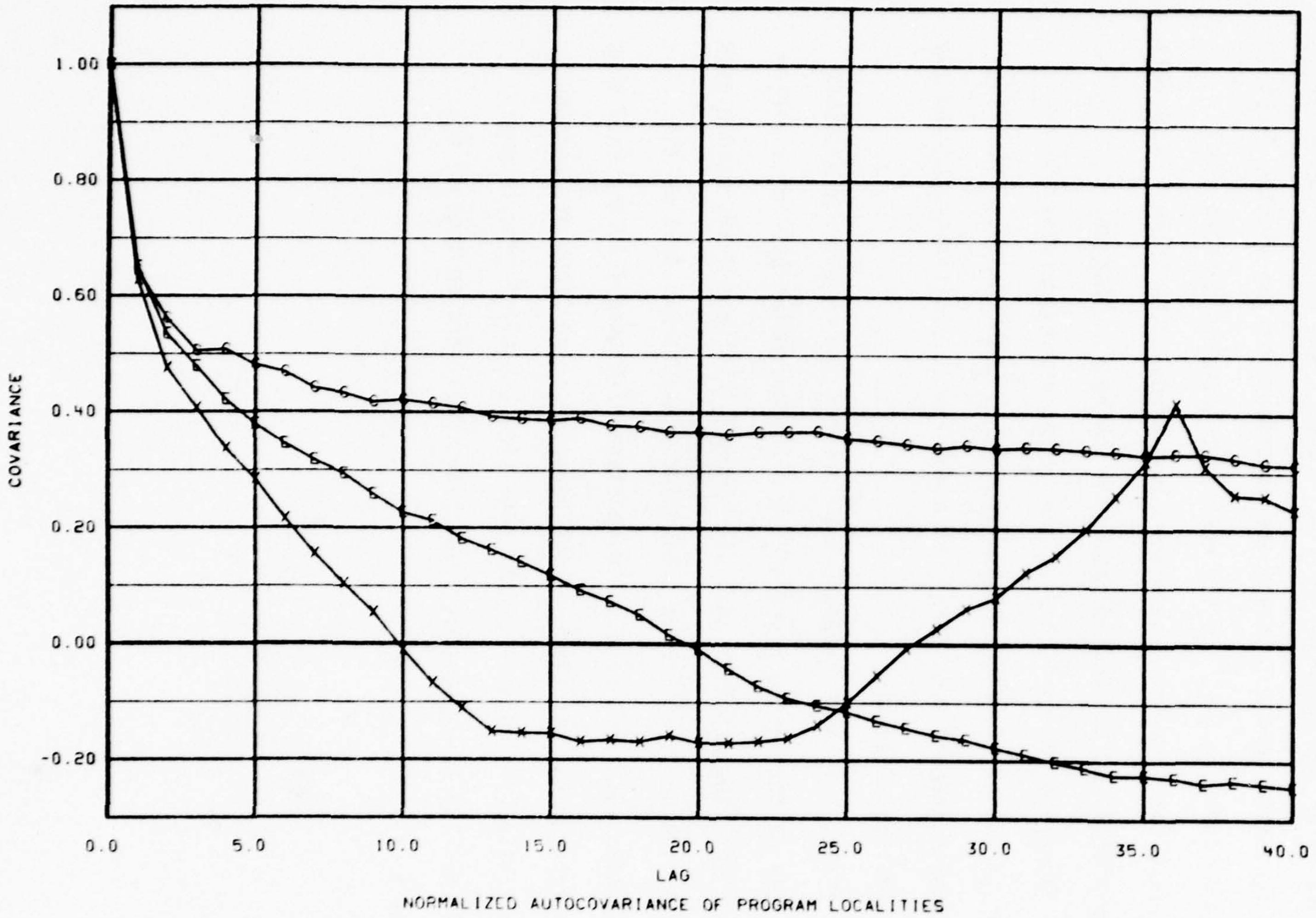


FIGURE 4-14 Additional Result for the MUDDLE Compiler

The next series of figures present results derived from the two traces of the MIDAS Assembler. Figure 4-15 and 4-16 give the mean localities and autocovariance functions for the shorter first trace. In Figure 4-16 one can observe a sizeable peak in the covariance at a lag of 27 for the large window case (Case E $\approx$ 16,384) again indicating the second pass over the source program.

Figures 4-17 and 4-18 give the corresponding results for the second trace of the MIDAS Assembler. Here, both the mean localities and the covariance functions show a remarkable similarity between the two different traces.

Finally, Figures 4-19 and 4-20 present the results of a second analysis of the second trace of the MIDAS Assembler but for a larger page size. The original analysis (Figures 4-17 and 4-18) used a page size of 128 words while the subsequent analysis was done on the basis of a 1024 word page. A comparison of Figure 4-18 and 4-20 show that the general structure of the covariance functions are changed very little by variation of the page size. Similar results were found to also hold for other page size variations on some of the other traces.

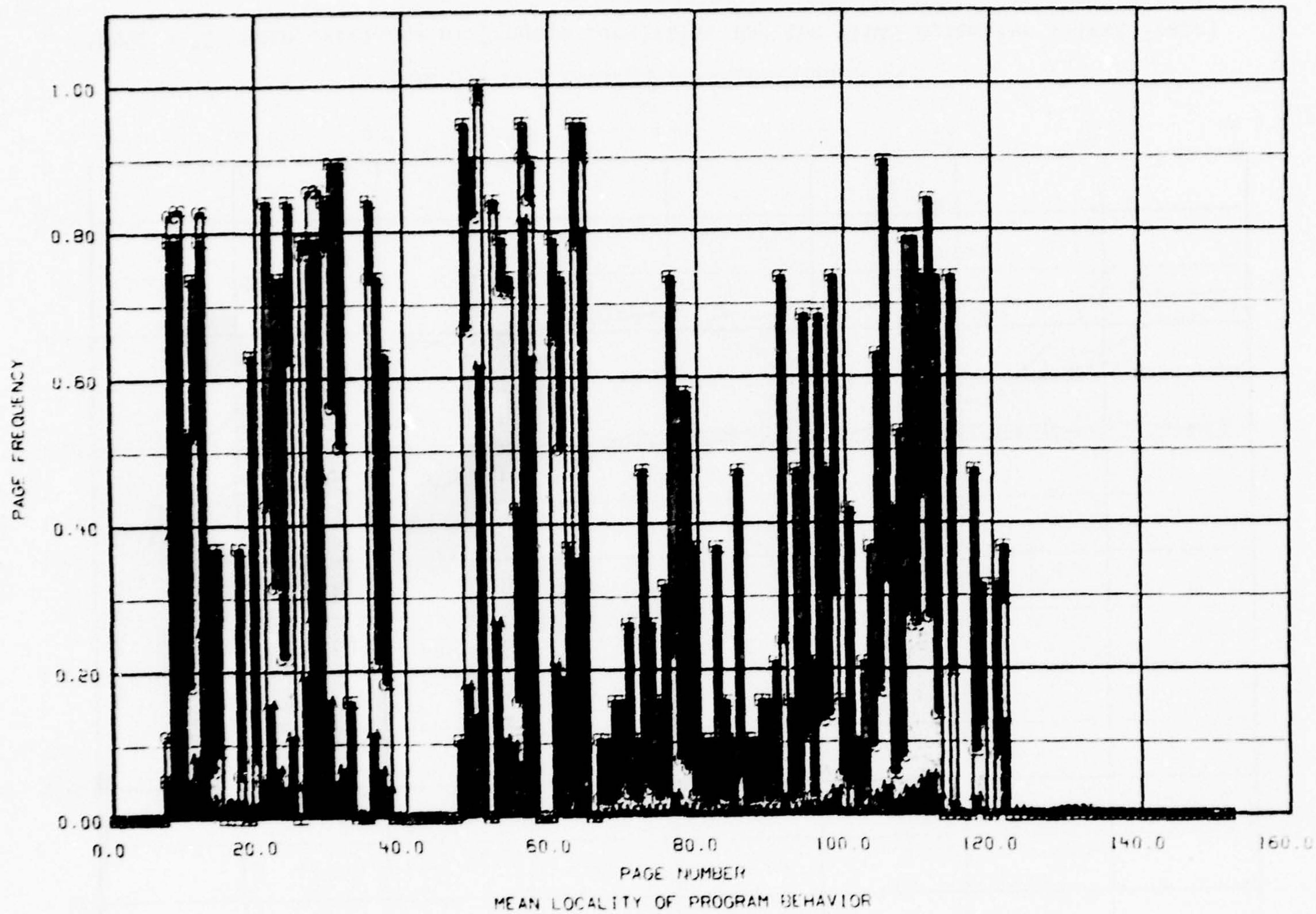


FIGURE 4-15 Mean Localities for the MIDAS Assembler (First Trace)

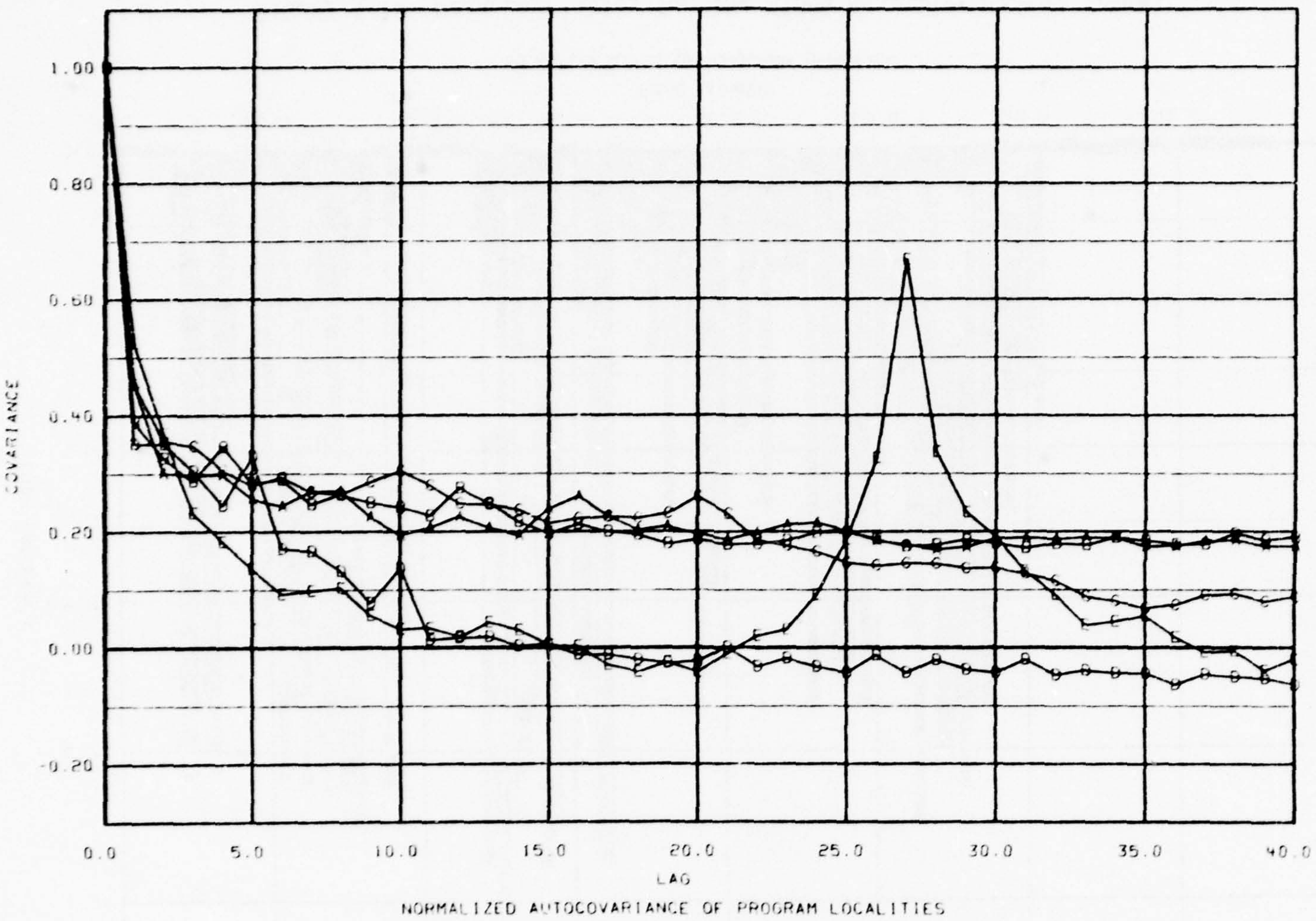


FIGURE 4-16 Autocovariance of Program Localities for the MIDAS Assembler (First Trace)

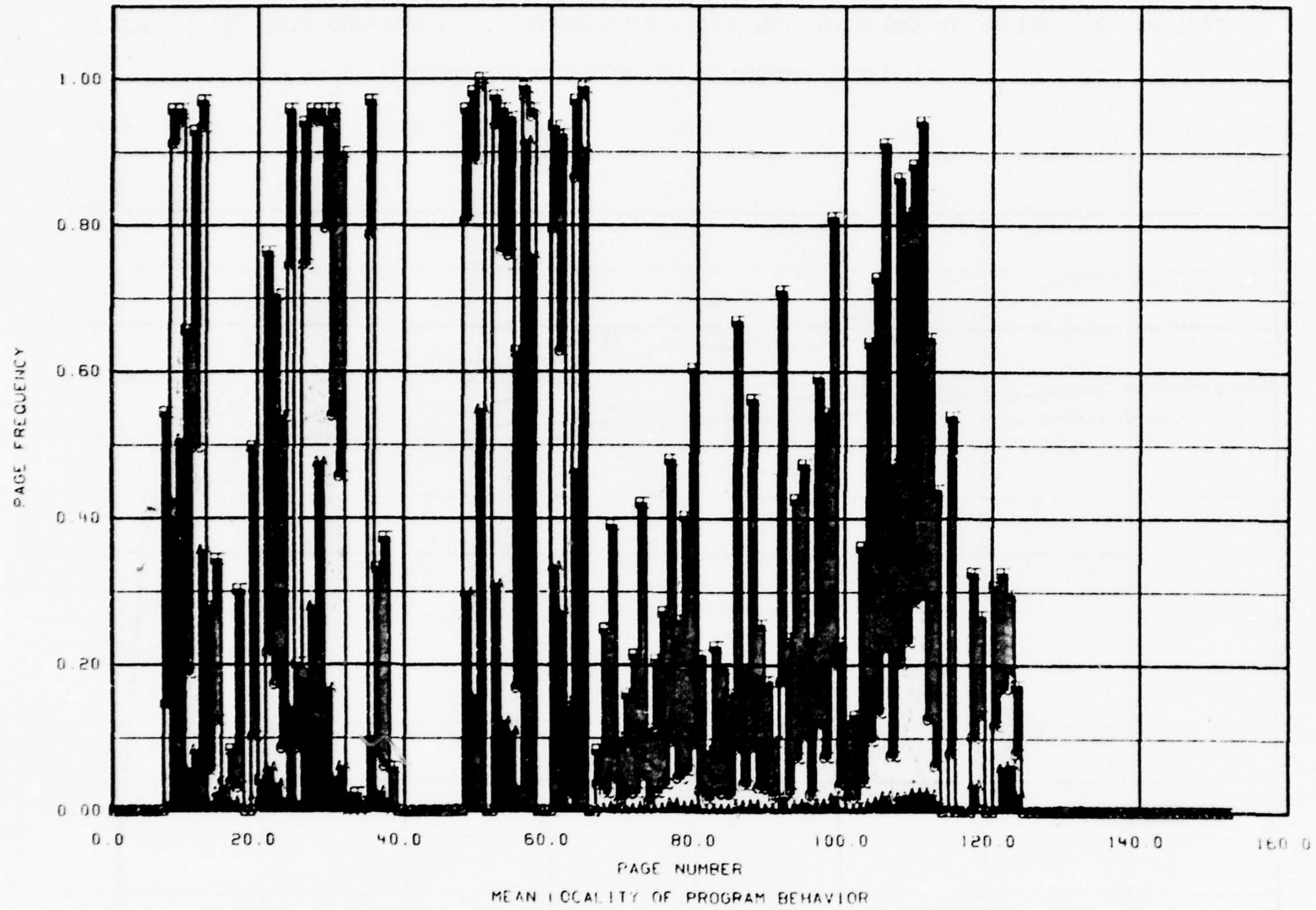


FIGURE 4-17 Mean Localities for the MIDAS Assembler (Second Trace)

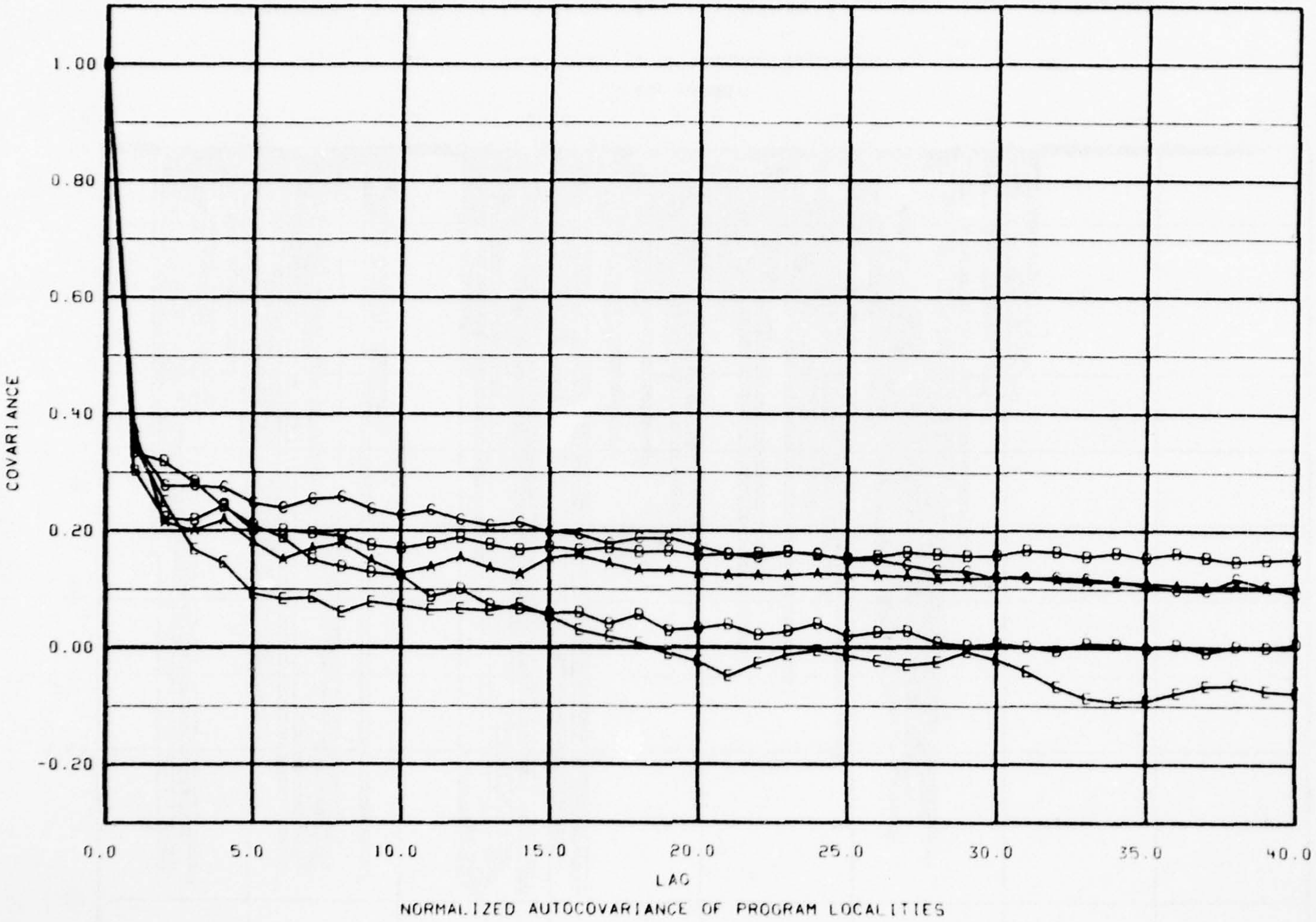


FIGURE 4-18 Autocovariance of Program Localities for the MIDAS Assembler (Second Trace)

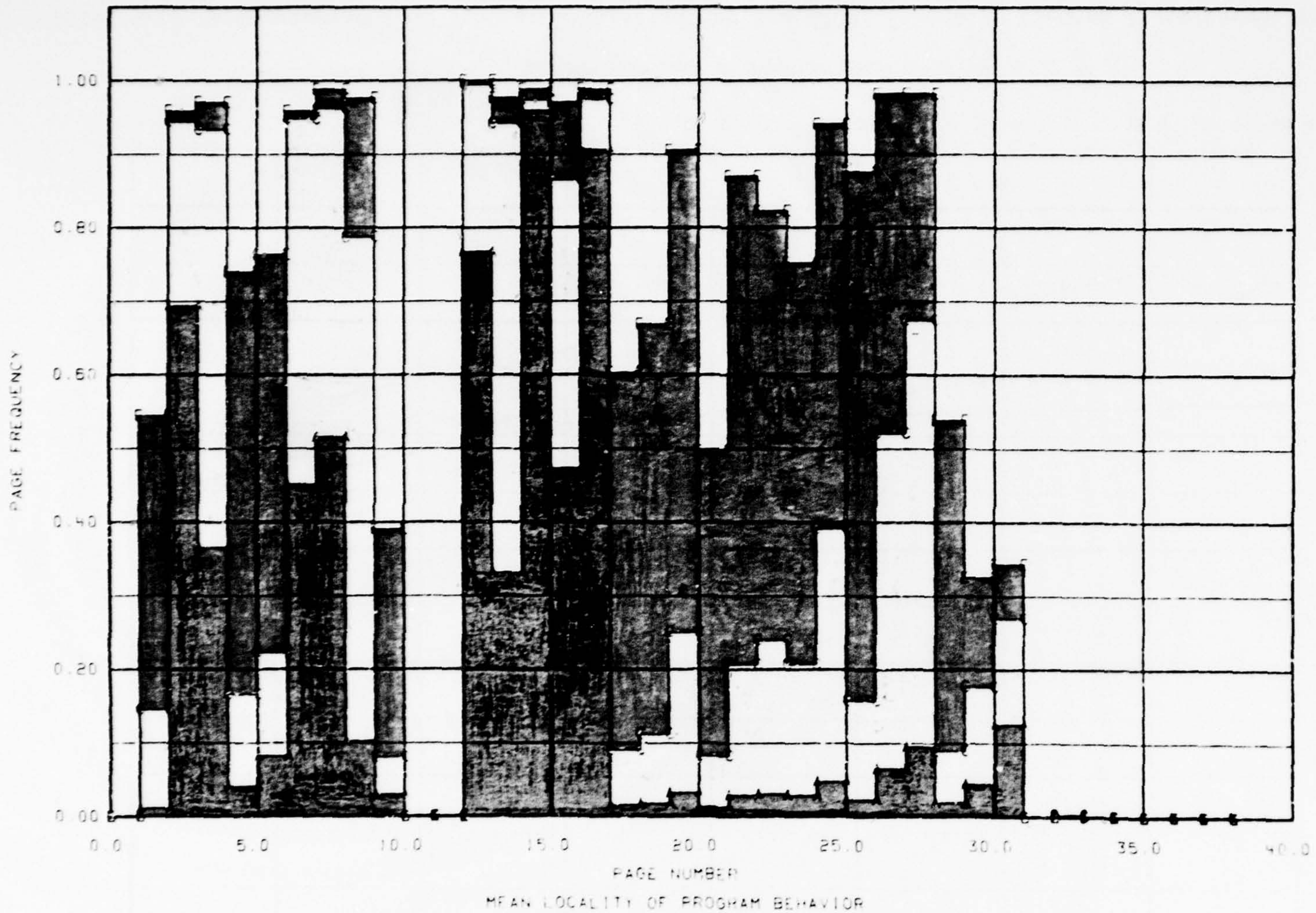


FIGURE 4-19 Mean Localities for the MIDAS Assembler (Second Trace/Large Page)

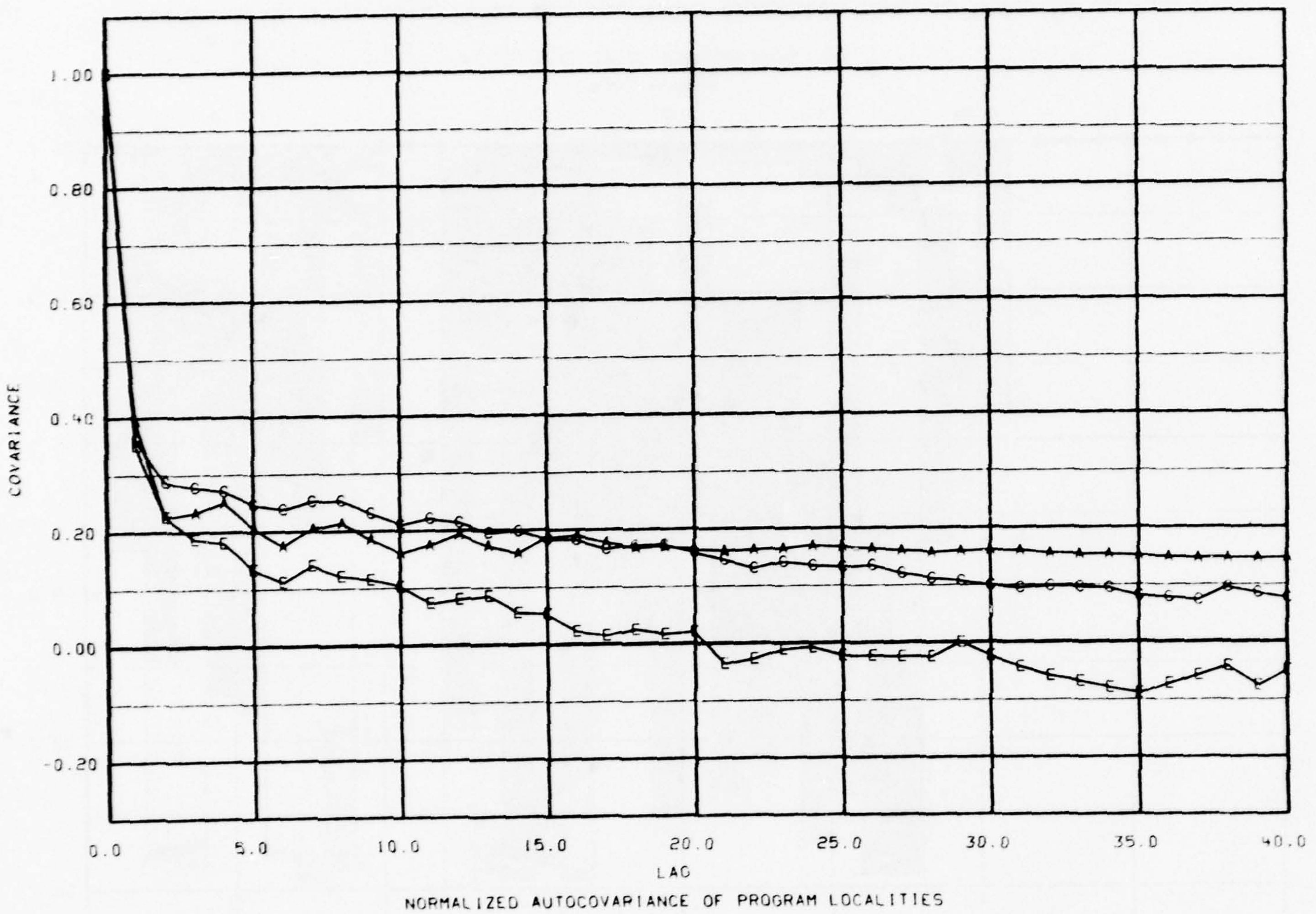


FIGURE 4-20 Autocovariance of Program Localities for the MIDAS Assembler (Second Trace/Large Page)

The final set of figures give results for the Text Editor, TECO; and the application program, VECTOR-SMOOTH.

Figures 4-21 and 4-22 present the mean localities and covariances for the text Editor, TECO, which was in the process of using macro commands to walk through the structures of a MUDDLE Program checking that it was properly parenthesized.

The application program, VECTOR-SMOOTH, which averaged a sequence of random floating-point numbers over a moving window, was a rather tightly looping program. Its expected behavior was observed. Figures 4-23 and 4-24 present three cases for window (interval) sizes of 256, 1024, and 4096. The tight looping of the program is exhibited in the periodic nature of the autocovariance functions.

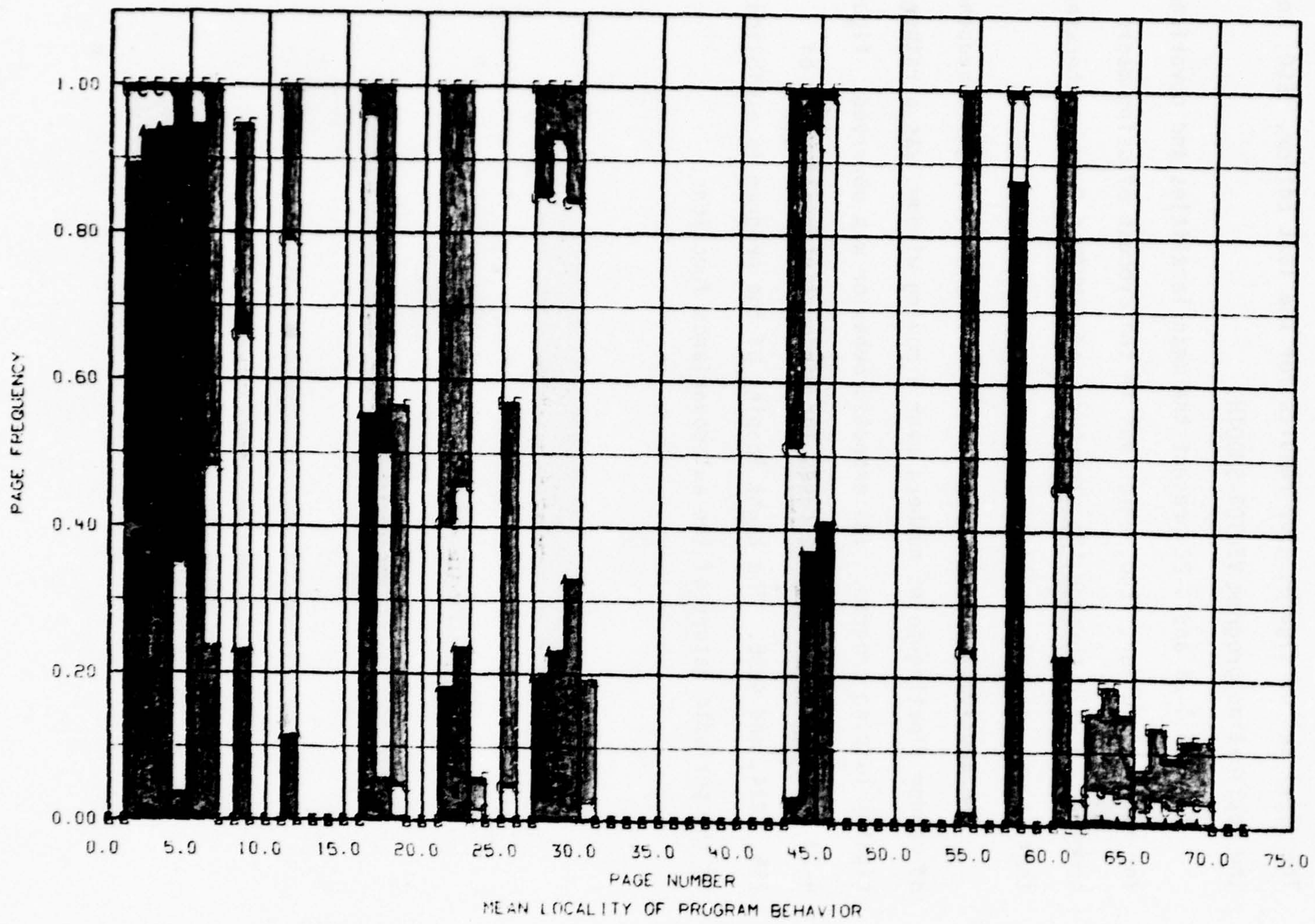


FIGURE 4-21 Mean Localities for the Text Editor TECO

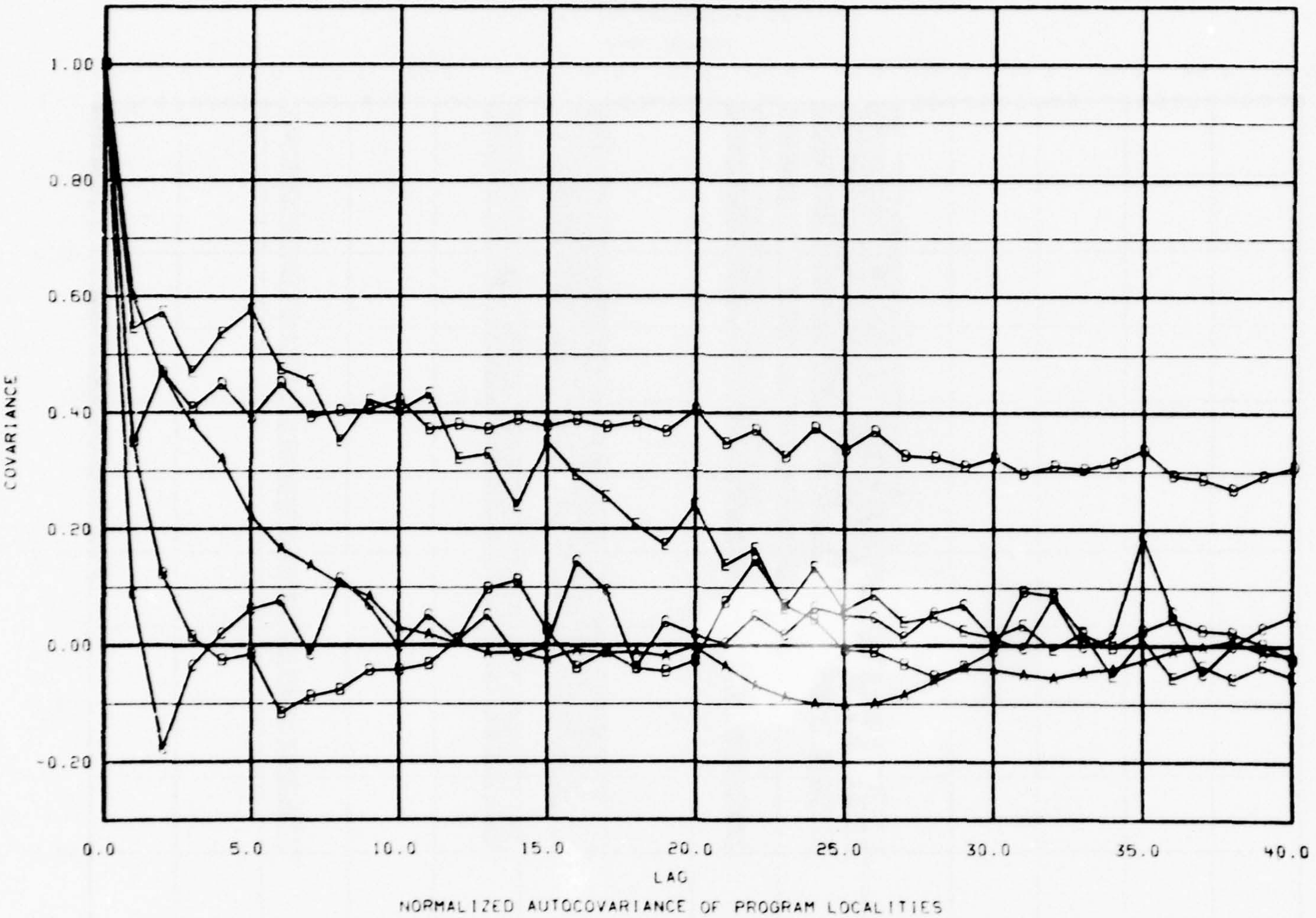


FIGURE 4-22 Autocovariance of Program Localities for the Text Editor TECO

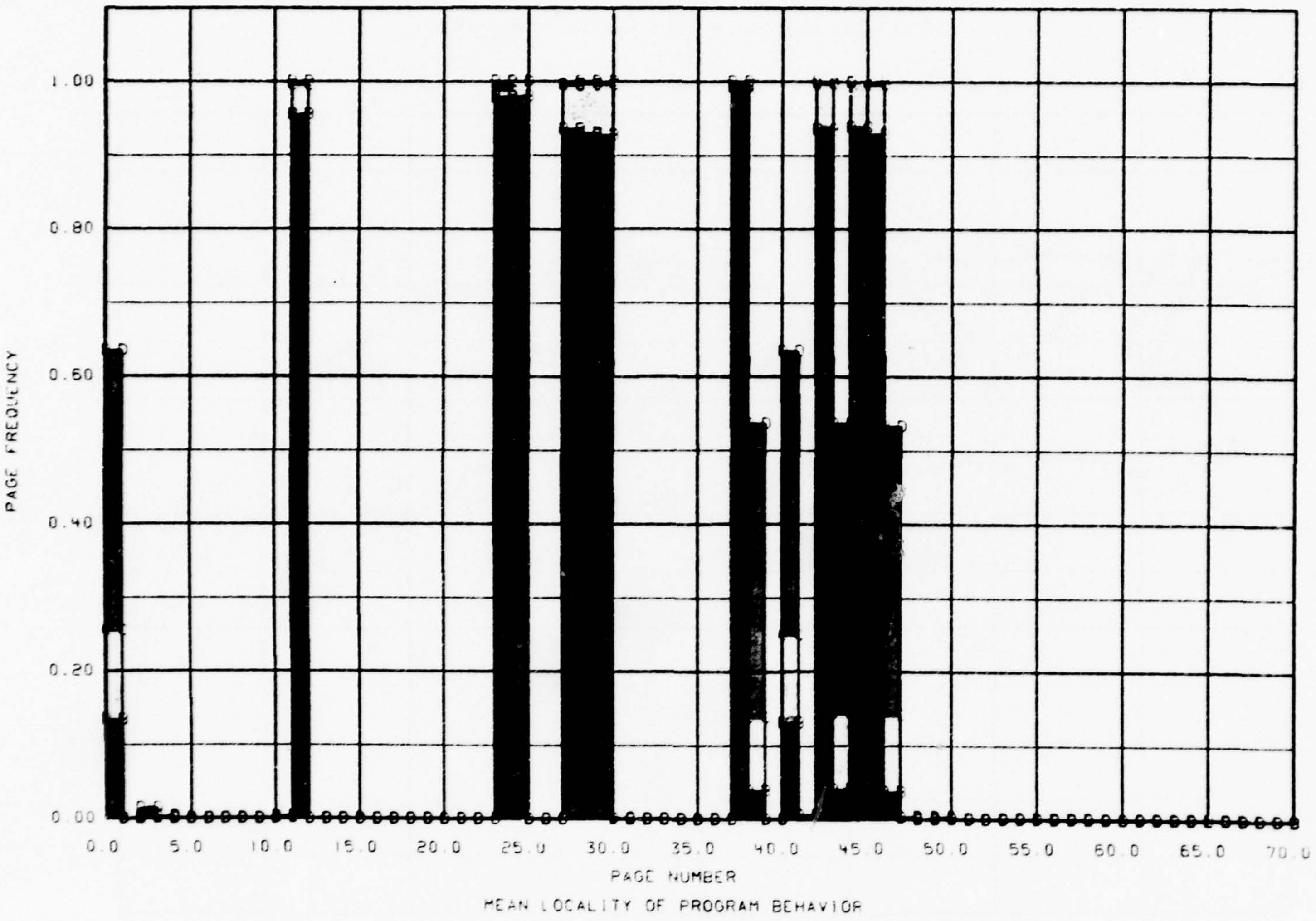


FIGURE 4-23 Mean Localities for the Application Program VECTOR-SMOOTH

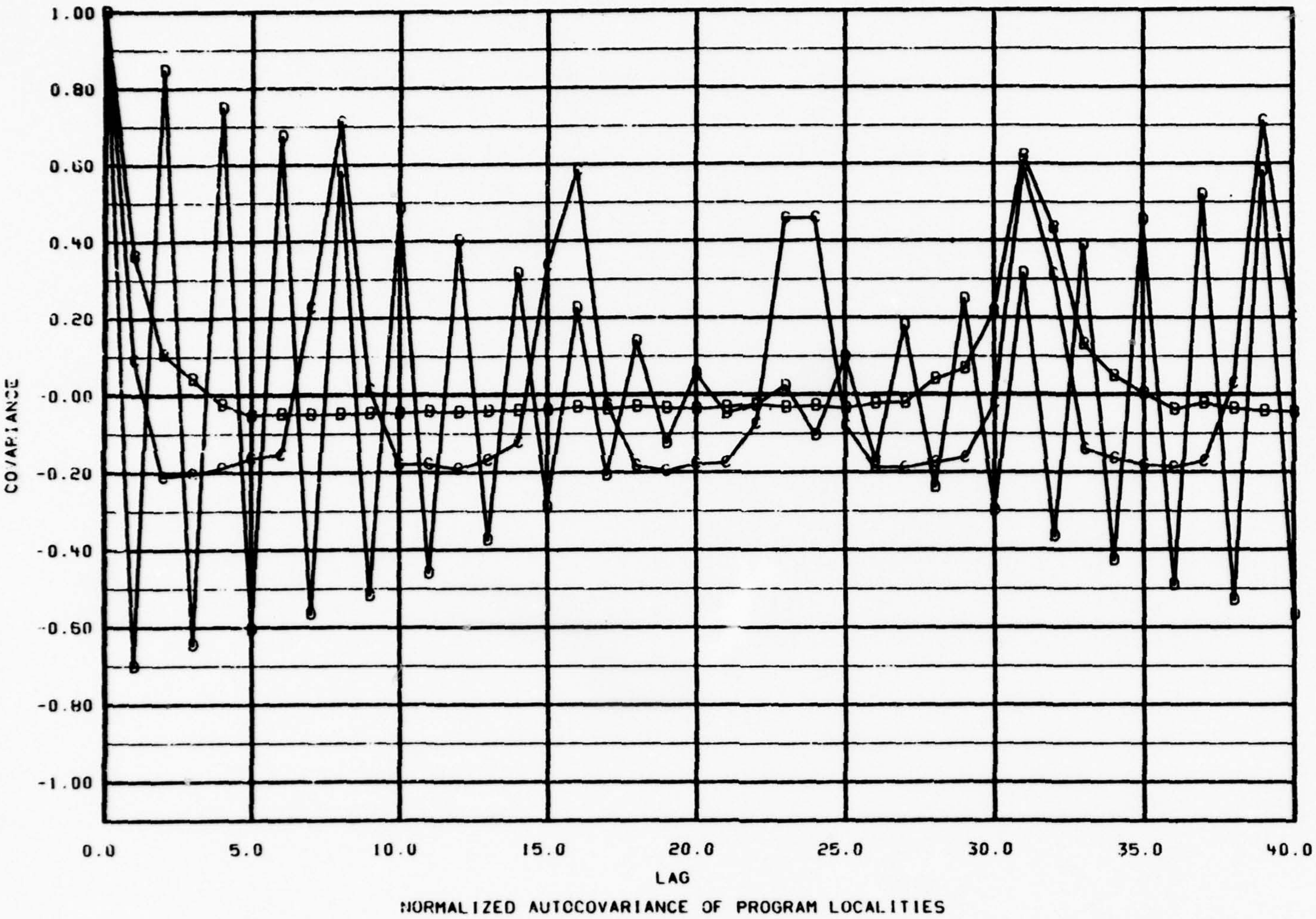


FIGURE 4-24 Autocovariance of Program Localities for the Application Program VECTOR SMOOTH

CHAPTER 5

A MODEL OF PROGRAM BEHAVIOR

5.1 The Need and Constraints for an Analytic Model

From the previous discussion, it is obvious that the problem of memory management is one of page replacement. As we have noted in Section 2.4.2, most previous paging algorithms can be derived from Denning's [Den70] informal "Principal of Optimality". In its statement, the Principle itself recognizes a dichotomy between those algorithms for which the future of the page reference string is known precisely and those for which it is not. In the known (deterministic) case, the Principle yields Belady's MIN algorithm or the Prieve-Fabry VMIN algorithm, respectively, depending upon whether the size of the memory partition is fixed or variable. When the future of the reference string is not known, the Principle requires one to use the conditional expectation (i.e., a prediction) of the future of page usage. Consequently, all practical memory management policies and their associated paging algorithms are fundamentally one of prediction.

In order to do prediction, one must have some statistical model for the memory usage behavior. There are many ways to introduce stochastic models for the page reference string. In turn, each different model will yield a different paging algorithm. Ideally, one seeks a model and an associated predictor which represents a compromise between the maximization of information content and a minimization of implementation complexity.

As a first constraint on our stochastic model for memory usage behavior, we require that the model give a general class (family) of predictors which can be applied to the execution of all programs. Specifically, we do not want the structure of the predictor to change depending upon which executing program it is working against. In order to get the structural stability of the predictor, we thus must also characterize all the various program behaviors within one class of stochastic models.

For our purposes, we have chosen to limit our predictors to the simpler linear ones for which design procedures are well developed. However, even among linear predictors, there is still a dichotomy between those which are time-invariant and those which are time-varying. Again in the interest of least implementation complexity, we choose the time-invariant ones. As such, the predictors can be predetermined and, moreover, will not require any up-date overhead.

Given that the predictor is to be a predetermined linear time-invariant one, it is obvious from a control-theoretic viewpoint that the integral formulation of the optimal predictor due to Wiener [Lee60] is more appropriate for our purposes than the differential formulation due to Kalman and Bucy [BuJ68]. Moreover, with the subject constraints upon the predictor, the Wiener theory requires only mean values and the second order statistics (autocovariances) as the maximally useful information. Additionally, the above constraint of structural stability of the predictor requires that we model the memory usage behavior of all programs by a single family of autocovariance behavior. Consequently, the one final pragmatic requirement upon the model of program behavior is that a single family of covariance models summarize all of the empirically observed behavior as reported in the last Chapter.

In summary, as a model for program behavior, we require a predetermined single class of analytic second order (covariance) stochastic models which summarize all of the observed behavior as presented in Chapter 4. It must be predetermined and analytic to allow prior design of the predictor. It must be of a single class to get the structural stability of the predictor. It must be of second-order (covariance) to get the maximal information content with the constraint of a linear time-invariant predictor. Finally, it must provide a reasonable approximation to the behavior of real programs.

5.2 The Model: Preliminaries

A considered review of all of the estimated autocovariance functions for program localities presented in the last chapter suggest that they all might be reasonably approximated by a sum of exponentials. Specifically, for $\tau \geq 0$, an approximation of the form

$$\phi_a(\tau) = C_1\exp(a_1\tau) + C_2\exp(a_2\tau) + \cdots + C_n\exp(a_n\tau) \tag{5.1}$$

is suggested or, equivalently, in the sampled data case by the form

$$\phi_a(\tau) = C_1\mu_1^\tau + C_2\mu_2^\tau + \cdots + C_n\mu_n^\tau \tag{5.2}$$

where

$$\mu_k = \exp(a_k) \ . \tag{5.3}$$

Moreover, except for the MUDDLE Assembler whose covariance exhibits a damped oscillatory behavior, the empirical results indicate that the approximation can be realized in terms of only real exponentials. In either case of whether the approximation is in terms of only real exponentials or the more general complex exponentials (i.e., the a_k's are real or complex) the exponential nature of the covariance functions yields the rational spectra required for the factorization technique of the Wiener filter theory.

Again, in the interest of least implementation complexity, we seek a minimal number, n, of terms in the above approximations. Additional study of the results of Chapter 4 suggests that most of the autocovariance functions exhibit two components in their behavior. The first being a short-term component consisting of a rapidly decaying real exponential or in the case of the MUDDLE Assembler, a complex conjugate pair of damped exponentials. The second component consists of a slowly decaying real exponential which is the result of the long-term behavior associated with phase transitions of the executing program. Consequently, the empirical estimates of

Chapter 4 suggest that only two or three terms in the above approximations should suffice.

Therefore, as a preliminary analytic model of program usage behavior, we have chosen a second-order stationary random process whose autocovariance is given by two or three terms of the above approximations (5.1) or (5.2). Specific models for predictor design require the determination of the amplitude coefficients C_k's and the exponents a_k's to which we now turn.

5.3 Prony's Method with Modifications

A classical technique of some nearly 200 years for fitting the above exponential approximations to empirical data is due to Prony [Hil56]. The technique itself is known as Prony's method and has an extensive history. Specifically, the method has a well known problem with limited precision or noisy data [Lan56] due to the general lack of orthogonality among the approximating complex exponential base functions. For our purposes, which requires only a few terms in the approximation and yet has sufficient data points to employ least-squares techniques to the over determined parts of solution technique, Prony's method has been modified to provide a satisfactory solution.

For specific details of Prony's method, we direct the reader to Hildebrand's treatment (see [Hil56] , pp. 378-382) which we have used. Here, we only present the method in its briefest outline so as to indicate its modifications for least-squares techniques.

Specifically, Prony's method assumes that one is given values of $\phi(\tau)$ at N equally spaced points $\tau = 0, 1, 2, \ldots, (N-1)$, say $\phi_0, \phi_1, \phi_2, \ldots, \phi_{N-1}$. Moreover, if the approximation $\phi_a(\tau)$ is to satisfy the observed values with equality, then the following system of equations must hold:

$$\begin{array}{lllllll} C_1 & + \; C_2 & + & \ldots & + \; C_n & = & \phi_0 \\ C_1 \mu_1 & + \; C_2\mu_2 & + & \ldots & + \; C_n\mu_n & = & \phi_1 \\ C_1 \mu_1^2 & + \; C_2\mu_2^2 & + & \ldots & + \; C_n\mu_n^2 & = & \phi_2 \\ \ldots & \ldots & & \ldots & & \ldots & \\ C_1 \mu_1^{N-1} & + \; C_2\mu_2^{N-1} & + & \ldots & + \; C_n\mu_n^{N-1} & = & \phi_{N-1} \end{array} \qquad (5.4)$$

However, before the linear system (5.4) in the unknown amplitudes C_k can be solved, the μ's must first be determined as the roots of an algebraic equation

$$\mu^n - \alpha_1\mu^{n-1} - \alpha_2\mu^{n-2} - \ldots - \alpha_{n-1}\mu - \alpha_n = 0, \qquad (5.5)$$

whose coefficients α's must even earlier be determined as the solution of the following system of (N-n) equations:

$$\begin{array}{lllllll}
\phi_{n-1}\,\alpha_1 & + & \phi_{n-2}\,\alpha_2 & + & \cdots & + \;\phi_0\,\alpha_n & = \phi_n \\
\phi_n\,\alpha_1 & + & \phi_{n-1}\,\alpha_2 & + & \cdots & + \;\phi_1\,\alpha_n & = \phi_{n+1} \\
\cdot\;\cdot & & \cdot\;\cdot & & \cdot\;\cdot\;\cdot & \cdot\;\cdot & \cdot\;\cdot\;\cdot \\
\phi_{N-2}\,\alpha_1 & + & \phi_{N-3}\,\alpha_2 & + & \cdots & + \;\phi_{N-n-1}\,\alpha_n & = \phi_{N-1}
\end{array} \tag{5.6}$$

Prony's method requires that N = 2n in order to exactly solve (5.6) for the n α's. For our purposes, n = 2 or 3 while N = 41. Consequently, the system (5.6) is greatly over-determined and we have employed a (real) generalized-inverse procedure [B-IG74] to yield a least-square approximation to the coefficients α's.

Having the α's, the μ's are next determined as the roots of (5.5). Finally, the system (5.4) can be solved for the amplitudes C_k. Again, in our case with N >> n, the sytem (5.4) is over-determined. Here, because the roots μ are in general complex, a complex generalized-inverse procedure must be used to provide a least-squares estimate of the C_k's.

5.4 An Approximation Program

A general purpose FORTRAN program has been written which implements Prony's method with the modifications indicated above. Specifically, the program accepts as inputs the following:

1) n: the number of terms in the approximation,
2) N: the number of sample values ϕ_k, and the
3) ϕ_k's: the sampled values.

The program then first establishes the system of equations (5.6) which it solves in a least-squares sense by means of a generalized inverse procedure for the coefficients ϕ_k. The α's are then used in a root finding subroutine to obtain the μ's. The μ's are then used to establish a system of equations (5.4) which is solved by the program by a complex generalized inverse procedure to yield the amplitudes C_k's. In the course of its execution, the program also tabulates the following:

1) the roots μ_k,
2) the exponents a_k, and
3) the amplitudes C_k.

Having determined a specific approximation (5.1), the program then evaluates it at values of its argument τ corresponding to the input data. The errors or residuals representing the differences between the original data points and the approximation are then calculated and tabulated. The residuals are also used to calculate and report a value for the root-mean-square (RMS) error. Finally, the program jointly plots the original data and the approximation to provide a rapid visual assessment of the degree of approximation realized by

Prony's method.

The correctness of the approximation program was first verified by applying it to several sets of synthetic data constructed from the composition of two, three, and four real and/or complex exponentials. In all cases, the progam returned very good approximations to the synthetic data.

5.5 Sample Results

The Prony approximation program was first applied for the case n = 2 (i.e., a two term approximation) to all set of values for the empirical covariance functions of Chapter 4. For a great majority of empirical data sets, the two-term Prony approximation was quite good. Figures 5-1 through 5-3 present three typical examples of the type of approximation realized. While the two-term approximation does not capture the oscillatory nature of the covariance function in the case of the MUDDLE Assembler (Fig. 5-2), it still gives a good gross approximation over both the short-term and the long-term behavior of the covariance process. Table 5-1 contain some of the specific two-term approximations realized for the autocovariances of the program localities with a window size of 1024 references (Case C).

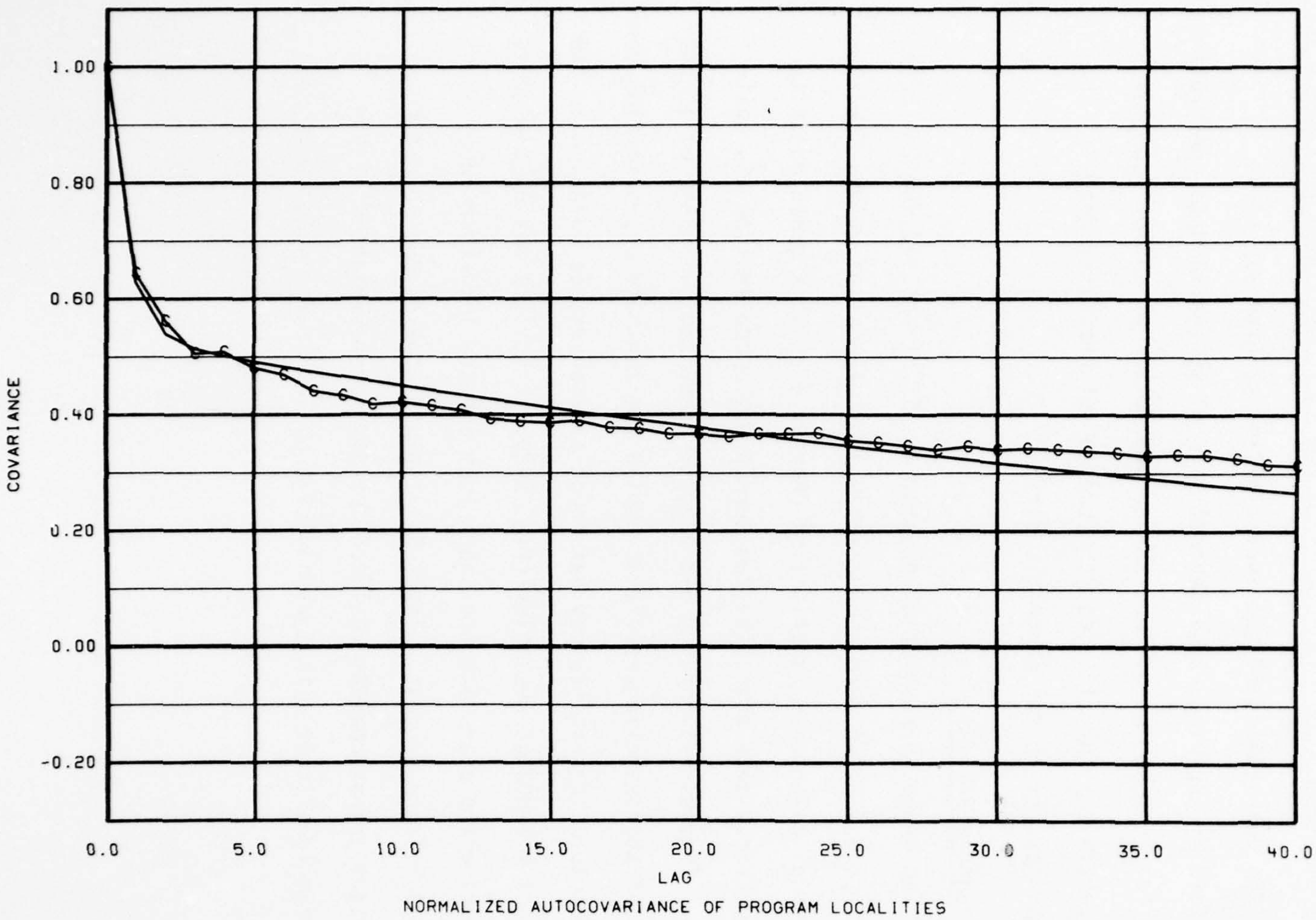


FIGURE 5-1 Two-Term Prony Approximation to an Autocovariance Function for the MUDDLE Compiler

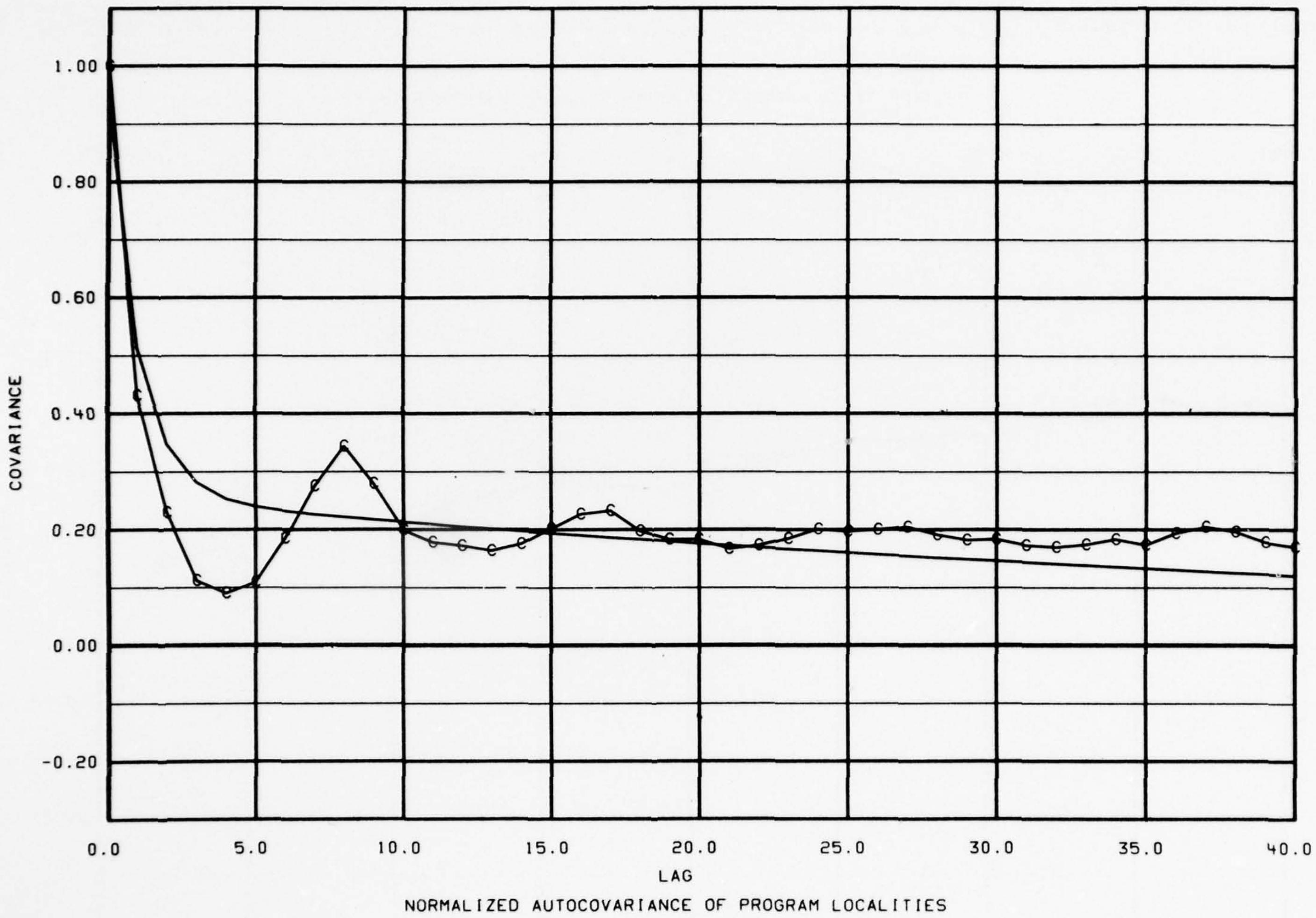


FIGURE 5-2 Two-Term Prony Approximation to an Autocovariance Function for the MUDDLE Assembler (First Trace)

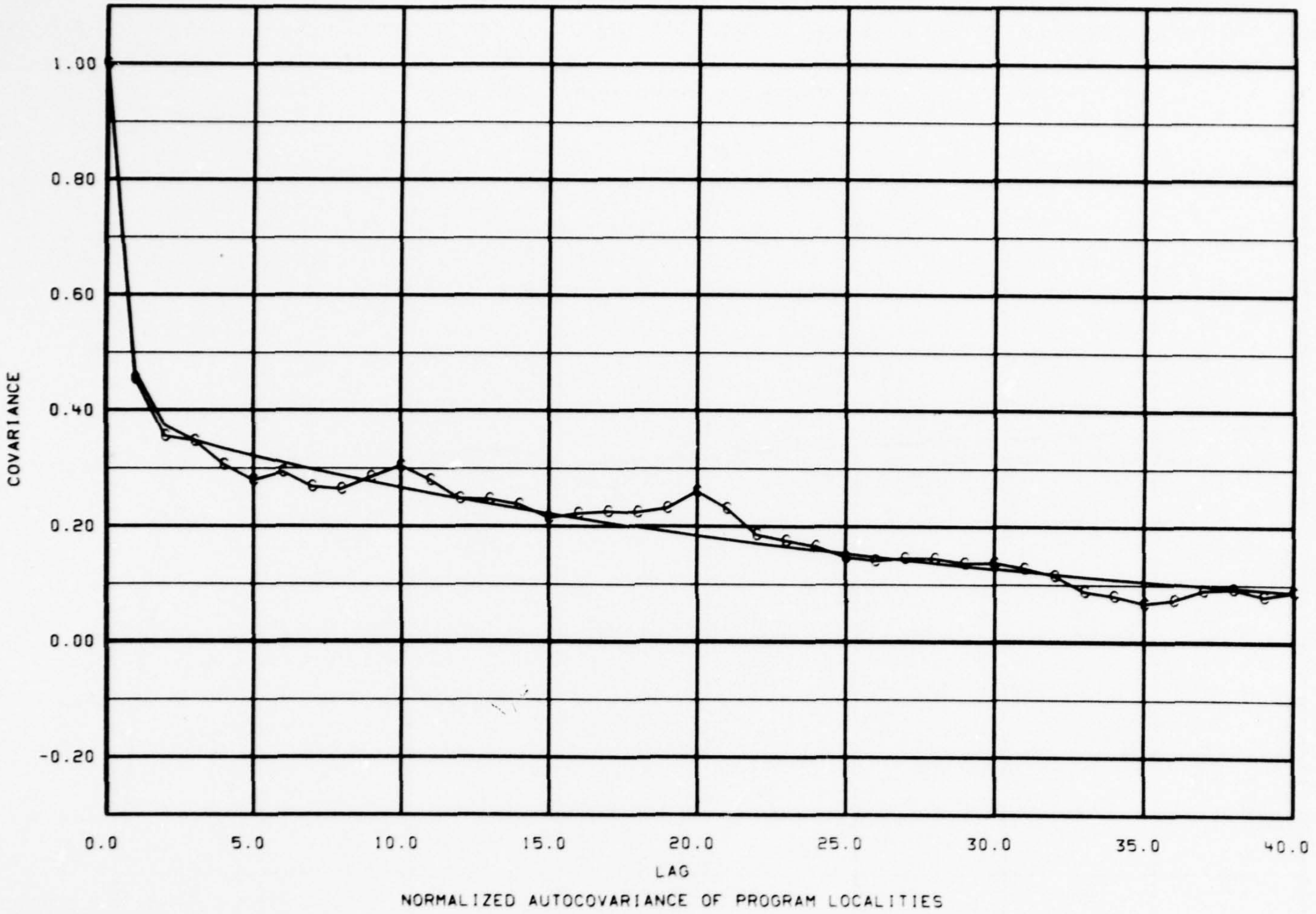


FIGURE 5-3 Two-Term Prony Approximation to an Autocovariance Function for the MIDAS Assembler (First Trace)

PROGRAM	APPROXIMATION
MUDDLE Compiler	$\phi_a(\tau) = 0.468 \exp(-1.52\|\tau\|) + 0.536 \exp(-0.018\|\tau\|)$
MUDDLE Assembler	
First Trace	$\phi_a(\tau) = 0.680 \exp(-0.958\|\tau\|) + 0.257 \exp(-0.019\|\tau\|)$
Second Trace	$\phi_a(\tau) = 0.730 \exp(-1.197\|\tau\|) + 0.250 \exp(-0.007\|\tau\|)$
MIDAS Assembler	
First Trace	$\phi_a(\tau) = 0.611 \exp(-1.852\|\tau\|) + 0.387 \exp(-0.037\|\tau\|)$
Second Trace	$\phi_a(\tau) = 0.701 \exp(-2.241\|\tau\|) + 0.298 \exp(-0.027\|\tau\|)$

TABLE 5-1 TWO-TERM PRONY APPROXIMATIONS

Primarily to obtain a better approximation in the case of the MUDDLE Assemble, the approximation program was re-executed on all data sets to obtain the three-term approximations. In general, the addition of another term to the approximation produced little to negative results. Figure 5-4 presents the three-term approximation corresponding to the two-term result for the MUDDLE Assembler of Figure 5-2. While it might not be obvious from a comparison of the two figures, a comparison of the RMS errors shows the two-term approximation to be better than the three-term one — 0.064 vs. 0.089. Moreover, the three-term approximation has also failed to capture the oscillatory nature of the empirical data. The specific approximation in this case being given by

$$\begin{aligned}\phi_a(\tau) = 1.992 \exp(-1.301|\tau|) - 1.363 \exp(-0.798|\tau|) + \\ + 0.399 \exp(-0.049|\tau|).\end{aligned} \tag{5.7}$$

Increasing the freedom of the Prony approximation to even four and five terms still failed to capture the oscillatory nature of the MUDDLE Assembler's autocovariance function. Moreover, in all cases where an approximation of more than two terms yield a better result, the increase or gain was of such little consequence so as not be justified by the corresponding increase in the model's complexity.

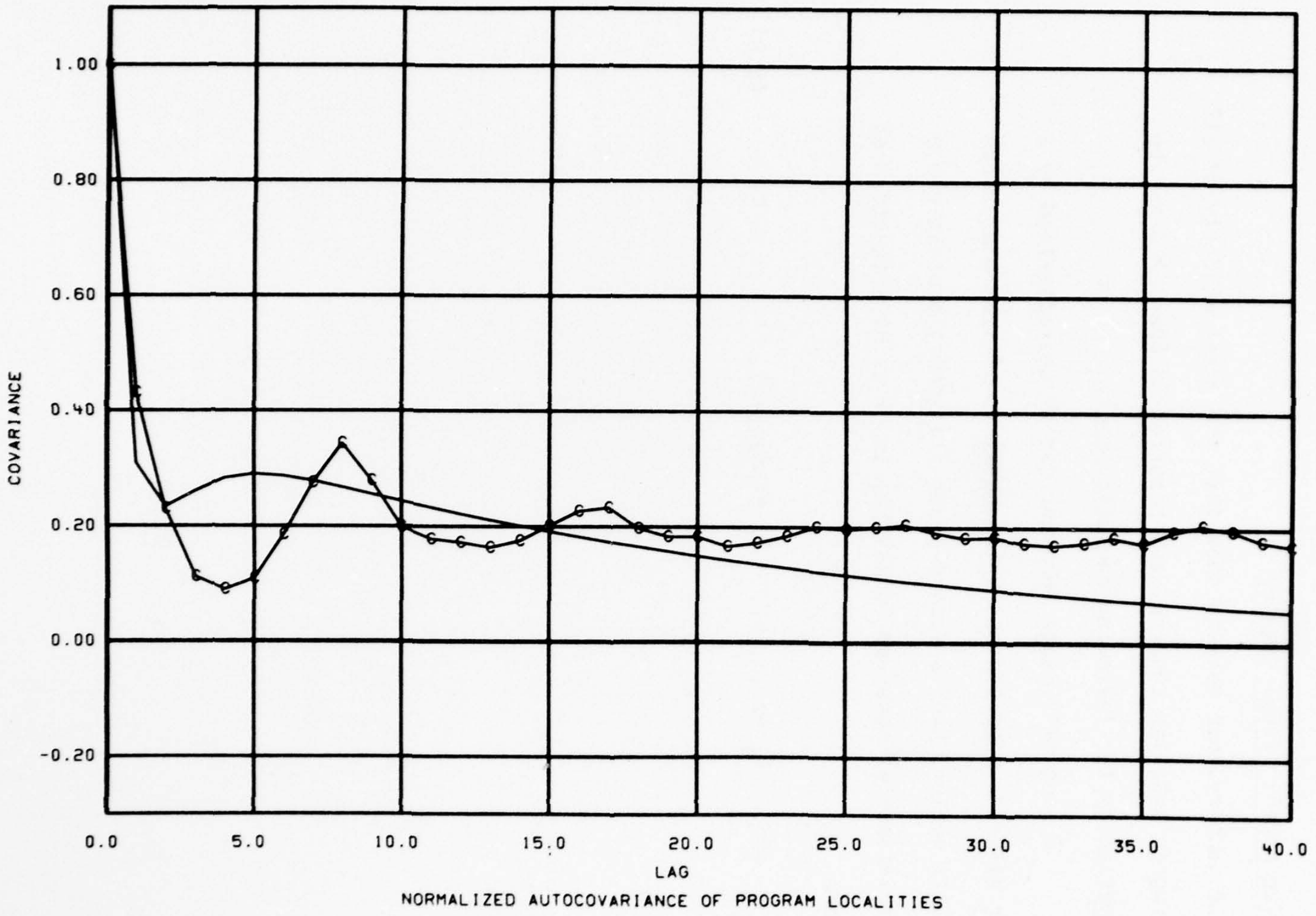


FIGURE 5-4 Three-Term Prony Approximation to an Autocovariance Function for the MUDDLE Assembler (First Trace)

5.6 The Model: Final Summary

In summary, our results show that the great majority of all the program behavior observed and reported in Chapter 4 can be encompassed by the following analytic model:

A program's usage behavior can be characterized by a vector valued random process which consists of some mean value (locality vector) plus a second-order stationary process whose autocovariance of observed localities is given by a function of the form

$$\phi(\tau) = C_1 \exp(a_1|\tau|) + C_2 \exp(a_2|\tau|). \qquad (5.8)$$

CHAPTER 6

A CONTROL-THEORETIC APPROACH TO MEMORY MANAGEMENT

6.1 Introduction and Problem Identification

How any specific program's memory demands are met is properly within the domain of the memory management portion of an operating system. Ideally, in each time interval, the operating system should provide in main memory precisely those pages required by each executing resident program. There are, however, several well known aspects of the memory management problem which preclude such an idealized solution. First and foremost, there is a real lack of reliable prior information about the specific memory demands each resident program will make upon the system. Secondly, the physical resource (primary memory) is of limited extent and may not be able to accommodate all of the required pages. There is also the matter of costs associated with the under-utilization of memory and/or the moving of a program's pages between primary memory and secondary storage. Because of these complications, the memory management function in large computer systems is a complex problem.

From a control-theoretic viewpoint, the memory management function is a constrained stochastic optimization problem. First, it is an optimization problem in that one generally tries to reduce the various costs associated with under-utilized equipment and/or job delays. Its stochastic nature is intrinsic and is reflected by cur lack of prior information of specific job demands. The problem's solution is also constrained by the finite extent of primary memory.

Fortunately, for our purposes, Aho, Denning and Ullman [ADU71] have shown that demand paging is optimal if the cost C(k) of bringing in k pages simultaneously, is not less than the cost of bringing them in individually (i.e., if $C(k) \geq k$). While some computer hardware can support the case where $C(k) < k$, the observed locality behavior of program address reference string does not support the arbitrary loading of even adjacent pages and thus precludes any such cost savings. Consequently, the case $C(k) = k$ is reasonable and the restriction of future considerations to demand paging algorithms is quite realistic.

With the restriction of the memory manager to a demand paging scheme, the only remaining freedom is in the choice of the page replacement algorithm. From the discussion of the previous chapter, all practical page replacement algorithms are fundamentally one of prediction (or estimation of the conditional expectation) of future page usage. Moreover, the determination of specific predictors or estimators is predicated upon some stochastic model and its associated probability distributions. Our choice of the specific stochastic model in Chapter 5 was determined by two principal considerations. First and foremost was the empirical results of Chapter 4. The second principal consideration was the pragmatic reality that any model and its associated predictor must represent a compromise between information content and implementation complexity. Our specific choice of a second-order stationary random process model was, consequently, motivated by a knowledge that it provided a minimal implementation complexity for a predictor via Wiener filter theory and yet required as maximal useful information, the means and covariance functions of Chapter 4.

The model of Chapter 5 also contains another significant feature which is very important to the implementation issue in practical operating systems. Namely, the model contains the macro-structure of memory usage rather than micro-structure of individual page usage as addressed in Jain's thesis [Jai78]. It is felt that the implementation of a different yet specific predictor for each page's usage will be unrealistic for some time into the future. We do, however, feel that future LSI technology can provide hardware support for the memory management function only if it is of a highly parallel nature. A high degree of parallelism translates into a requirement that all the individual page usage predictors be of the same structure and, at least within a given program execution, use the same coeffients (parameters). Consequently, we require the design of a single predictor algorithm which is to be uniformly applied to all pages of a given program. The design constraint of uniformity of the predictors requires that we use some average behavior of the individual page usages (see below). The reciprocity results of Section 4.4.3, guarantee that the autocovariances of our program locality model are equivalent to the required average of the individual page autocovariance functions.

In summary, our control-theoretic yet pragmatic view of the memory management function is realized in the design of a single optimal Wiener filter to be implemented and used uniformly across all pages of an executing program as a predictor of future usage of each individual page. The next section gives a rigorous formulation and solution of such a Wiener filter problem.

6.2 The Wiener Pure Predictor

In this section, we formulate and solve for a *single optimal* Wiener pure predictor to be used uniformily across all page usage histories.

6.2.1 Problem Formulation. From the results of the two previous chapters, the evolution of program's memory usage during execution can be modeled as a vector-valued random process of observed localities,

$$\underset{\sim}{L}(i) = \left(\beta_1(i),\ \beta_2(i),\ \ldots,\ \beta_n(i),\ \ldots,\ \beta_N(i)\right) . \qquad (6.1)$$

The $\underset{\sim}{L}(i)$ process has a mean value given by the mean locality vector,

$$\underset{\sim}{\overline{L}} = \left(\overline{\beta}_1,\ \overline{\beta}_2,\ \ldots,\ \overline{\beta}_n,\ \ldots,\ \overline{\beta}_N\right) \qquad (6.2)$$

and an autocovariance $\phi(j)$ as defined by equation (4.19) which is realized by a function of the form

$$\phi(j) = C_1\exp(a_1|j|) + C_2\exp(a_2|j|). \qquad (6.3)$$

If we view the vector usage history $\underset{\sim}{L}(i)$ component-wise as an ensemble of *individual page usage* histories $\beta_n(i)$, then by the reciprocity result of Section 4.4.3, the function (6.3) also provides the ensemble average

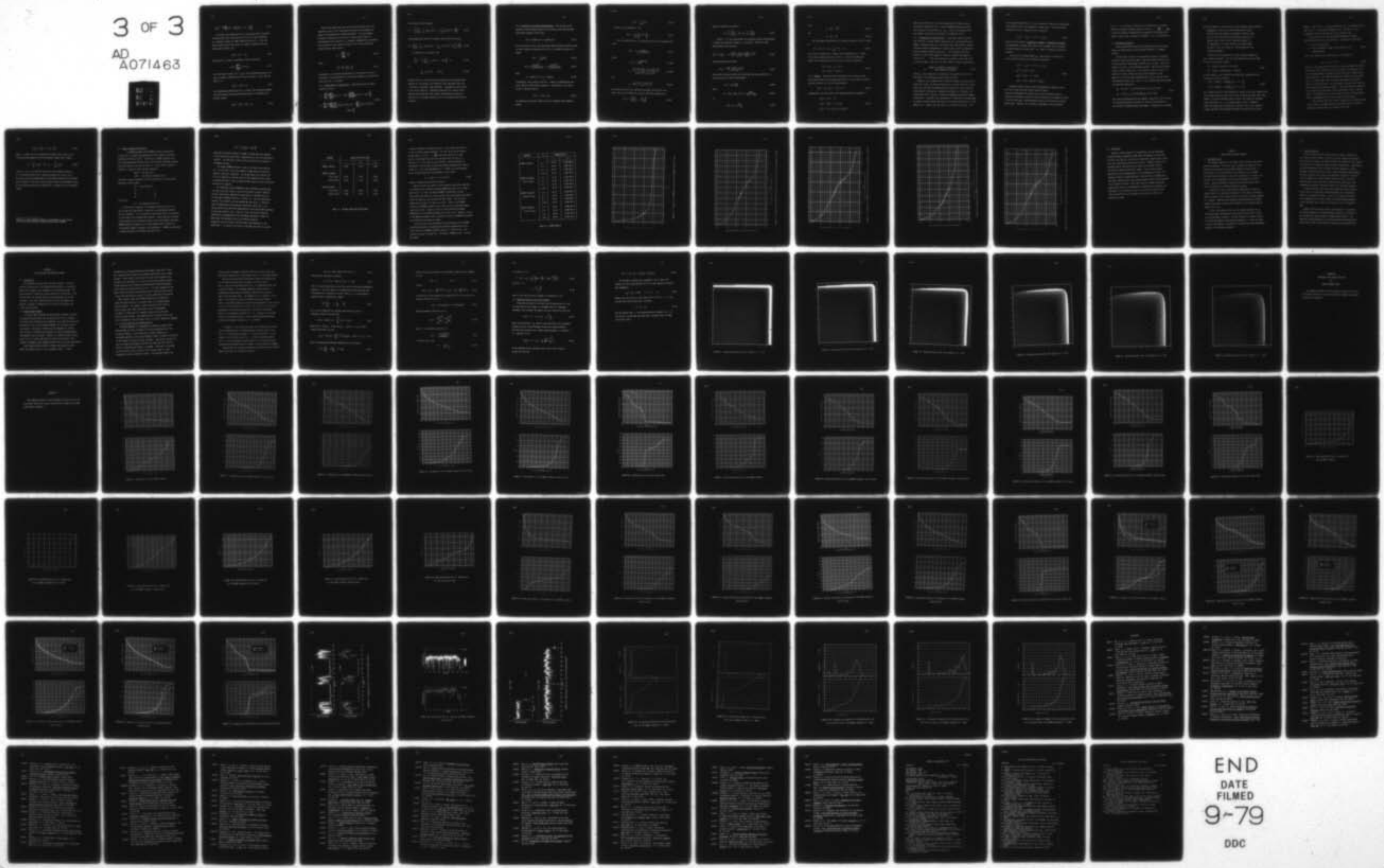
$$\psi(j) = \frac{1}{N}\sum_{n=1}^{N} \psi_n(j) \qquad (6.4)$$

of the individual page autocovariance functions

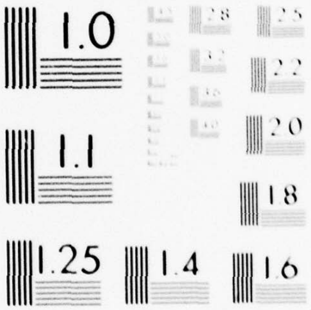
AD-A071 463 NAVAL UNDERWATER SYSTEMS CENTER NEW LONDON CT NEW LO--ETC F/G 9/2
OPTIMIZATION OF COMPUTER OPERATING SYSTEMS.(U)
APR 79 C R ARNOLD

UNCLASSIFIED NUSC-TR-6045 NL

3 OF 3
AD
A071463



END
DATE
FILMED
9-79
DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

$$\psi_n(j) = \underset{i}{\text{Ave}}\left\{\left[\beta_n(i) - \overline{\beta}_n\right]\left[\beta_n(i + j) - \overline{\beta}_n\right]\right\}. \qquad (6.5)$$

We require the determination of a single physically realizable discrete-time linear time-invariant filter to be used uniformily as a predictor across all page usage histories $\beta_n(i)$. Since the page histories have a mean value, we require a predictor for the non-constant portion, say

$$x_n(i) = \beta_n(i) - \overline{\beta}_n. \qquad (6.6)$$

Specifically, we require a predictor filter of the form

$$y(i) = \sum_{j=0}^{\infty} h_j\, x(i-j) \qquad (6.7)$$

such that when it inputs $x(i) = x_n(i)$, the corresponding output, say $y_n(i)$, provides a prediction of the $x_n(i)$ process α units into the future, i.e.,

$$y_n(i) = \hat{x}_n(i + \alpha). \qquad (6.8)$$

The instantaneous prediction error is clearly the difference between the actual output and the desired output which is realized by the process, namely,

$$e_n(i) = y_n(i) - x_n(i + \alpha). \qquad (6.9)$$

While exact predictions upon the stochastic process $x_n(i)$ are impossible, one can still make predictions which are optimal with respect to some measure of system performance. For our purposes, we require a measure of error which is always positive for any instantaneous error, and which is, moreover, mathematically tractable. Such a measure is the system wide total of all of the individual prediction mean square errors,

$$E_T = \sum_{n=1}^{N} \overline{e_n^2} \tag{6.10}$$

where

$$\overline{e_n^2} = \underset{i}{\text{Ave}} \left\{ e_n^2\,(i) \right\} . \tag{6.11}$$

Consequently, the ultimate determination of the predictor filter is contained in the determination of the filter weights h_j such that the total error E_T is a minimum.

6.2.2 Minimization of System Error. Substituting from the above equation, we have

$$E_T = \sum_{n=1}^{N} \underset{i}{\text{Ave}} \left(\sum_{j=0}^{\infty} h_j x_n(i-j) - x_n(i+\alpha) \right) \left(\sum_{k=0}^{\infty} h_k x_n(i-k) - x_n(i+\alpha) \right)$$

$$E_T = \sum_{n=1}^{N} \underset{i}{\text{Ave}} \left\{ \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} h_j h_k x_n(i-j) x_n(i-k) - 2 \sum_{k=0}^{\infty} h_k x_n(i-k) x_n(i+\alpha) + x_n^2(i+\alpha) \right\} \tag{6.12}$$

Performing the time averages,

$$E_T = \sum_{n=1}^{N} \left\{ \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} h_j h_k \, \psi_n(j-k) - 2 \sum_{k=0}^{\infty} h_k \psi_n(k+\alpha) + \overline{x_n^2} \right\} . \tag{6.13}$$

Interchanging the order of summation, and using (6.4) we have

$$E_T = N \left\{ \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} h_j h_k \, \psi(j-k) - 2 \sum_{k=0}^{\infty} h_k \, \psi(k+\alpha) + \frac{1}{N} \sum_{n=1}^{N} \overline{x_n^2} \right\} . \tag{6.14}$$

To minimize E_T, we require that

$$\frac{\partial E_T}{\partial h_k} = 2N \left\{ \sum_{j=0}^{\infty} h_j \, \psi(k-j) - \psi(k+\alpha) \right\} = 0 \tag{6.15}$$

or

$$\sum_{j=0}^{\infty} h_j \, \psi(k-j) = \psi(k+\alpha) . \tag{6.16}$$

Equation (6.16) is the discrete-time equivalent of the Wiener-Hopf equation for the optimal pure predictor (the continuous time case is treated in [Lee60] , pages 406-409). Consequently, the solution of our filter problem is completely equivalent to a Wiener filter problem except that the usual process autocorrelation function is replaced by the ensemble average of all of the page autocorrelation functions.

6.2.3 Solution of the Wiener-Hopf Equation. The solution of our predictor filter problem requires the solution of the discrete-time Wiener-Hopf equation (6.16) where

$$\psi(j) = C_1\exp(a_1|j|) = C_2\exp(a_2|j|). \tag{6.17}$$

For our particular case, the associated spectral density function given by the bi-lateral z-transform of (6.17) is a rational function of z, namely,

$$\Psi(z) = \sum_{j=-\infty}^{\infty} \psi(j)z^j \tag{6.18}$$

$$\Psi(z) = \frac{C_1(1-\mu_1^2)}{(1-\mu_1 z)(1-\mu_1/z)} + \frac{C_2(1-\mu_2^2)}{(1-\mu_2 z)(1-\mu_2/z)} \tag{6.19}$$

where

$$\mu_1 = \exp(a_1), \text{ and } \mu_2 = \exp(a_2). \tag{6.20}$$

Consequently, the solution of (6.16) is readily accomplished by the usual spectral factorization technique. Specifically, if we write (6.19) in factoral form as

$$\Psi(z) = \Psi^+(z)\ \Psi^-(z), \tag{6.21}$$

the generating (transfer) function for our required filter weights, namely,

$$H(z) = \sum_{j=0}^{\infty} h_j z^j \tag{6.22}$$

is given by (see [Whi63], p. 32)

$$H(z) = \frac{1}{\psi^+(z)} \left[z^{-\alpha} \psi^+(z) \right]_+ \quad . \tag{6.23}$$

For our particular case, the spectrum (6.19) can be factored such that

$$\psi^+(z) = G \frac{(1-cz)}{(1-\mu_1 z)(1-\mu_2 z)} \tag{6.24}$$

where

$$c = \frac{1}{2} b - \frac{1}{2} \sqrt{b^2 - 4} \quad , \tag{6.25a}$$

$$b = \frac{C_1(1-\mu_1^2)(1+\mu_2^2) + C_2(1-\mu_2^2)(1+\mu_1^2)}{C_1\mu_2(1-\mu_1^2) + C_2\mu_1(1-\mu_2^2)} \quad , \tag{6.25b}$$

and

$$G = \frac{C_1\mu_2(1-\mu_1^2) + C_2\mu_1(1-\mu_2^2)}{c} \tag{6.25c}$$

The portion (6.24) of the spectrum associated with positive time values can also be written by a partial fractions expression as

$$\psi^+(z) = G \left\{ \frac{R_1}{1-\mu_1 z} + \frac{R_2}{1-\mu_2 z} \right\} \tag{6.26}$$

where the residues are given by

$$R_1 = \frac{1 - c/\mu_1}{1 - \mu_2/\mu_1} \text{ , and } R_2 = \frac{1 - c/\mu_2}{1 - \mu_1/\mu_2} \ . \tag{6.27}$$

Finally, it is not unreasonable for operating systems considerations to choose the prediction interval α to be unity. With this final specialization, we can write

$$\left[z^{-1} \ \Psi^{+}(z)\right]_{+} = G \frac{\mu_1 R_1(1 - \mu_2 z) + \mu_2 R_2(1 - \mu_1 z)}{(1 - \mu_1 z)\ (1 - \mu_2 z)} \tag{6.28}$$

and substituting into (6.23),

$$H(z) = \frac{\mu_1 R_1 + \mu_2 R_2 - \mu_1 \mu_2 z}{(1 - cz)} \tag{6.29}$$

The transfer function (6.29) for the optimal one-step predictor can also be written in the following forms,

$$H(z) = g \, \frac{1 - dz}{1 - cz} \tag{6.30}$$

where

$$g = \mu_1 R_1 + \mu_2 R_2 \quad \text{and} \quad d = \frac{\mu_1 \mu_2}{\mu_1 R_1 + \mu_2 R_2} \tag{6.31}$$

or

$$H(z) = G_1 + \frac{G_2}{1 - cz} \tag{6.32}$$

where

$$G_1 = g\frac{d}{c} = \frac{\mu_1\mu_2}{c} \tag{6.33a}$$

and

$$G_2 = g(1 - \frac{d}{c}) \tag{6.33b}$$

The time domain realization of our optimal predictor (6.32) is given by

$$y(i) = G_1\ x(i) + G_2 \sum_{j=0}^{\infty} c^j x(i - j), \tag{6.34}$$

which can be recognized as a simple gain (attenuator) plus a simple first order smoothing (low-pass) filter. In fact, the predictors output can be generated recursively by the system of equations,

$$\begin{aligned} Y(i) &= cY(i - 1) + x(i), \\ y(i) &= G_1x(i) + G_2Y(i). \end{aligned} \tag{6.35}$$

6.2.4 Summary. Recalling that the predictor y(i) is only for the non-constant portion of the page usage history, we can realize a specific one-step predictor for the n-th page usage as

$$z_n(i) = \bar{\beta}_n + y_n(i) = \hat{\beta}_n(i + 1). \tag{6.36}$$

Consequently, the final form of the required prediction equation is

$$\begin{aligned} x_n(i) &= \beta_n(i) - \bar{\beta}_n \\ Y_n(i) &= cY_n(i - 1) + x_n(i) \\ z_n(i) &= \bar{\beta}_n + G_1\ x_n(i) + G_2Y_n(i) \end{aligned} \tag{6.37}$$

where the coefficient c is given by equation (6.25) and G_1 and G_2 are given by equation (6.33). A small FORTRAN program was written which accepts the coefficients C_1, C_2, a_1 and a_2 of equation (6.3) and produces the required coefficients of equations (6.37).

6.2.5 Implementation Considerations. Using some scheme to provide time intervals of some nominal duration (e.g., by instruction count, address reference count, or a real-time clock), the optimal predictor algorithm could be implemented in software upon current systems which provide hardware supported usage-bits. For each page, one could maintain a count of the number of intervals in which the usage bit had been set. Then the estimate of $\bar{\beta}_n$ would be simply the ratio of the n-th page's count over the total interval count as expressed by

$$\bar{\beta}_n = \frac{\text{Number of Intervals containing } P_n}{\text{Total Number of Intervals}} \quad . \qquad (6.38)$$

Having $\bar{\beta}_n$, the computation or up-dating of the predictor (6.37) for each page of the executing programs would be direct.

However, since programs have a finite yet highly variable duration within the system, the above estimation of $\bar{\beta}_n$ requires that the various counts be reset when the program begins execution and to be saved and restored when the program is swapped out and back in. Moreover, the required division of (6.38) is often more demanding than other computer instructions. Therefore, the estimation of $\bar{\beta}$ might best be accomplished by another technique which does not use division and which uses a single memory cell — rather than two counts — to maintain its value. The usual engineering solution is to use a low-pass filter with a sufficiently long response time as an estimator of a mean value. The discrete-time equiavlent of such a low-pass filter is given by

$$v_n(i) = \lambda v_n(i - 1) + (1 - \lambda)\, \beta_n(i) \tag{6.39}$$

which is often called an exponential smoother or exponential estimator. The exponential estimator does require but one memory cell or register per page and does run "open-loop", i.e., does not require the reset of counters.

Since $v_n(i)$ will approximation $\overline{\beta}_n$, substitution of (6.39) into (6.37) yields the final set of prediction equations,

$$\begin{aligned} v_n(i) &= \lambda v_n(i - 1) + (1 - \lambda)\, \beta_n(i) \\ x_n(i) &= \beta_n(i) - v_n(i) \\ Y_n(i) &= cY_n(i - 1) + x_n(i) \\ z_n(i) &= v_i(n) + G_1 x_n(i) + G_2 Y_n(i) \end{aligned} \tag{6.40}$$

Although equation (6.40) could be implemented in software, their real merit stems from their highly parallel nature.
Future LSI technology will require very little in hardware costs to provide the continuous up-date of several hundred sets of equations (6.40) — one for each page. Microprogramming could also provide for variable coefficients. Morever, by a judicious adjustment of the coefficient

(e.g., λ, c, G_1 and G_2) so that they are expressible in a minimal fashion as composite negative powers of two (e.g., $\lambda = \frac{127}{128} = 1 - \frac{1}{128}$), a practical hardware implementation of equations (6.40) might only require simple shifts, and adds or subtracts — i.e., no multipliers.

6.3 The Memory Controller

Having established the prediction algorithms (6.40) for memory usage, our determination of the matching controller is somewhat informal.

If there were no errors in the prediction process, the memory controller should obviously keep or load those pages for which $z_n(i)$ is one and dismiss those pages for which $z_n(i)$ is zero. There will, however, be errors in the prediction process. While the true future values of page's usage $\beta_n(i + 1)$ can only be a one or a zero, our prediction filters (6.40) will only yield fractional values. However, by selecting some threshold value, say T_H, one can return the predicted values $z_n(i) = \hat{\beta}_n(i + 1)$ to a binary decision by making the following comparisons:

$$\begin{aligned} &D_0: \text{ If } z_n(i) < T_H, \text{ decide page } p_n \text{ will not be used.} \\ &D_1: \text{ If } z_n(i) \geq T_H, \text{ decide page } p_n \text{ will be used.} \end{aligned} \tag{6.41}$$

Even with the threshold decision, however, there will still be errors in the prediction-decision process so one must also provide for the loading of requested pages upon demand. Consequently, our memory controller reduces to the following variable (memory) space algorithm.

Controller At the start of each observation interval for each executing program, the memory controller should keep or load those pages whose predicted usage $z_n(i)$ is greater than or equal to the threshold parameter T_H. Correspondingly, it should release any resident pages for which $z_n(i)$ falls below T_H. Missing pages will be loaded upon demand.

The value of the threshold T_H is thus a design parameter for such a memory controller. For any given threshold value, two types of errors are possible, namely,

$$\begin{aligned} &E_1\text{: A page is kept but is not used.} \\ &E_2\text{: A page is released but is required.} \end{aligned} \qquad (6.43)$$

For each page p_n, one can define the conditional probabilities for each type of error as follows:

$$\begin{aligned} P_\alpha(n) &= \text{Prob}(E_1) = \text{Prob}\left[D_1 | \beta_n(i+1) = 0\right] \\ P_\beta(n) &= \text{Prob}(E_2) = \text{Prob}\left[D_0 | \beta_n(i+1) = 1\right] \end{aligned} \qquad (6.44)$$

By a suitable choice of the threshold T_H, either of the error probabilities can be made arbitrarily small at the cost of making the other very large. However, since the different types of errors are related to different system resources (i.e., $E_1 \sim$ primary memory, and $E_2 \sim$ secondary memory and supporting I/O processors, channels, etc.), the choice of T_H must represent some compromise between the effective use of main memory — small $P_\alpha(n)$'s — and the maintenance of a low page-fault-rate — small $P_\beta(n)$'s. The trade-off between the P_α's and the P_β's must consequently reflect some measure of their relative costs.

Prieve and Fabry in their development [PrF76] of the VMIN algorithm present similar considerations. Specifically, they define

R = Cost of a page-fault.

U = Cost of keeping one page in main memory for one reference time. (6.45)

and a cost function over a page trace of n references as

$$C(n) = n\ f\ R + n\ M\ U \tag{6.46}$$

where f is the page-fault-rate and M is the average memory utilization. Prieve-Fabry also recognize that by varying U/R, a performance curve in the average-memory-size/page-fault-rate plane can be generated for their algorithm. Their VMIN algorithm does minimize the cost (6.46), but it is unrealizable since it requires knowledge of the future portion of the page trace. For our purposes, the paper by Prieve and Fabry contains two important concepts. The first being the comparison of variable-space algorithms by their performance curves in the M-f plane. The second one is the use of the ratio U/R of the associated costs to parameterize the iso-cost curves in the M-f plane.

If we associate the costs U and R, respectively, with our type E_1 and E_2 errors, we recognize that our threshold value T_H should be choosen to minimize some cost function of the form

$$C(T_H) = K \left\{ P_\alpha \tau U + P_\beta \tau R \right\} \tag{6.47}$$

where τ is the size of the observation interval, and P_α and P_β are the system wide composite of the individual probabilities, namely,

$$P_\alpha = \sum_{n=1}^{N} P_\alpha(n) \quad \text{and} \quad P_\beta = \sum_{n=1}^{N} P_\beta(n). \tag{6.48}$$

Since P_α and P_β are implicit functions of the threshold value T_H, it's variation provides for a trade-off between the U and R costs. We do not pursue the determination of the optimal threshold any further.* We do, however, in the next section present sample M-f performance curves for our predictor-controller algorithm for a parametric set of threshold values.

*There is a strong analogy between our development in this Section and the detection problem of communications theory [Se165].

6.4 Sample Implementation Results

A FORTRAN program called OPTMEM has been written which provides for a simulated implementation of the predictor-controller algorithms (6.40) and (6.42). Specifically, OPTMEM simulates only one executing program and thus, any of the previous individual program trace data tapes can be used to drive it. As with earlier analysis programs, it requires input control parameters.

PGSIZE - the page size,

ISIZE - the interval (window) size T.

Additional required parameters are the coefficients of the predictor algorithms (6.40), namely

CL - the coefficient λ

CC - " " c

GI - " " G_1

G2 - " " G_2

and finally

TH - the threshold value T_H.

As with earlier programs, the parameters PGSIZE and ISIZE are used to map the input address string into a corresponding observed locality sequence. A set of prediction algorithms (6.40) is maintained for each page of the program under analysis. Based upon the occurrence of page faults and the simulated actions of the thresholded controller, OPTMEM maintains a dynamic list of page numbers which simulate the instantaneous memory assignments and requirements. OPTMEM also maintains a running estimate for the memory partition size

$$w(i) = \frac{1}{2} \left\{ N_S(i) + N_F(i) \right\} \tag{6.49}$$

where $N_S(i)$ and $N_F(i)$ denote the number of pages that are resident at the start and at the finish, respectively, of the i-th observation interval. The individual values of w(i) are saved in an array for subsequent analysis.

As output, OPTMEM provides a value of the fault-rate function f*( ) as the ratio of the total number of page faults to the total number of addresses referenced. An average memory utilization size $\overline{w}$ is also reported as the mean value of the sequence of w(i) values. A standard deviation σ and an autocorrelation $\phi_w(k)$ are also calculated from the w(i) sequence.

The simulation program OPTMEM has been repeatedly executed upon the same trace data tapes as used and described in earlier chapters. We have consistently employed the same page size (PGSIZE) for each of the sample programs traces (c.f. Table 3.1). For the observation interval size (ISIZE), we have settled upon the value T = 1024 and have correspondingly used predictor coefficient values for c, G_1, and G_2 derived from the Prony approximation to the associated covariance for the same interval size. Table 6-1 gives the optimal predictor coefficients corresponding to the covariance functions of Table 5-1. Other than the threshold value T_H for which we present parameteric sets of results, the only other free parameter is the coefficient λ in the set of filters (6.39) which are used to provide

PROGRAM	PREDICTOR COEFFICIENTS		
	c	G_1	G_2
MUDDLE Compiler	0.852	0.252	0.097
MUDDLE Assembler			
First Trace	0.922	0.409	0.035
Second Trace	0.952	0.315	0.028
MIDAS Assembler			
First Trace	0.829	0.182	0.109
Second Trace	0.849	0.115	0.103

TABLE 6-1 OPTIMAL PREDICTOR COEFFICIENTS

a running estimate of the mean locality $\bar{L}$. Our results show that it is not a critical design parameter. For the results we do present, we have used $\lambda = 0.99$ — $\lambda = \frac{127}{128}$ would have been just as good. We also investigated the start-up transient effect on the $v_n(\ )$ filters (i.e., set of zero or $\bar{\beta}_n$ initially). With essentially no difference in the results, we have consistently employed zero initial values (i.e., no prior knowledge of $\bar{L}$). Finally, for the threshold value, we have found that the parametric set of values

$$T_H = 0.05,\ 0.1,\ 0.3,\ \text{and}\ 0.5 \qquad (6.50)$$

produce a significant spectrum of results.

Table 6-2 gives the results of our predictor-controller algorithm as realized by the simulation program OPTMEM upon four of our sample data traces. In order to compare OPTMEM's performance, we have also plotted the Table 6-2 values opposite the corresponding performance curves for Denning's pure working-set (WS) scheme. The resulting comparisons are given in Figure 6-1 to 6-4. Clearly, the OPTMEM results on the MUDDLE Compiler trace (Fig. 6-1) are indistinguishable from those for WS. Likewise, there is little difference between OPTMEM and WS on the MIDAS Assembler trace (Fig. 6-4). However, on both of the MUDDLE Assembler traces, (Figures 6-2 and 6-3) OPTMEM consistently out performs the working-set scheme.

In order to get some assessment of the robustness of the OPTMEM predictor-controller, we employed the predictor determined from the first trace of the MUDDLE Assembler against its second trace. The results are given in Figure 6-5. Here again, OPTMEM out performed the WS scheme.

PROGRAM	T_H	OPTMEM RESULTS	
		$\overline{w}$	f*( )
MUDDLE Compiler	0.5	12.07	0.003 084 3
	0.3	14.82	0.001 803 1
	0.1	20.13	0.000 866 1
	0.05	23.47	0.000 576 5
MUDDLE Assembler	0.5	25.43	0.005 018 0
(First Trace)	0.3	29.16	0.003 275 2
	0.1	37.02	0.001 020 9
	0.05	41.36	0.000 457 3
MUDDLE Assembler	0.5	26.23	0.004 838 0
(Second Trace)	0.3	32.03	0.002 605 7
	0.1	39.41	0.000 882 5
	0.05	43.74	0.000 333 5
MIDAS Assembler	0.5	25.02	0.006 342 6
(First Trace)	0.3	30.78	0.003 831 0
	0.1	42.60	0.001 819 8
	0.05	50.54	0.001 116 3

TABLE 6-2 OPTMEM RESULTS

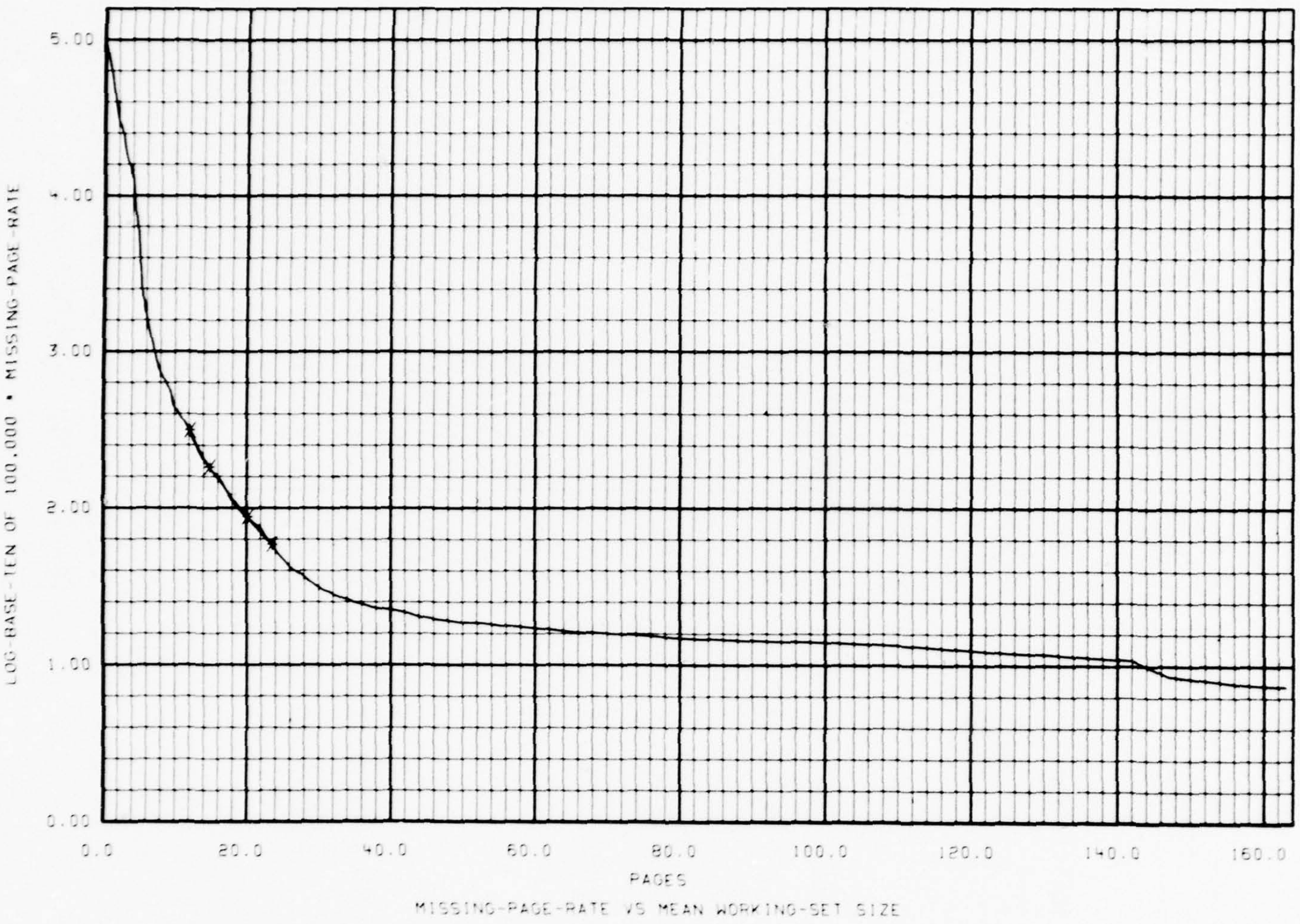


FIGURE 6-1 Comparison of OPTMEM to Pure WS for the MUDDLE Compiler Trace

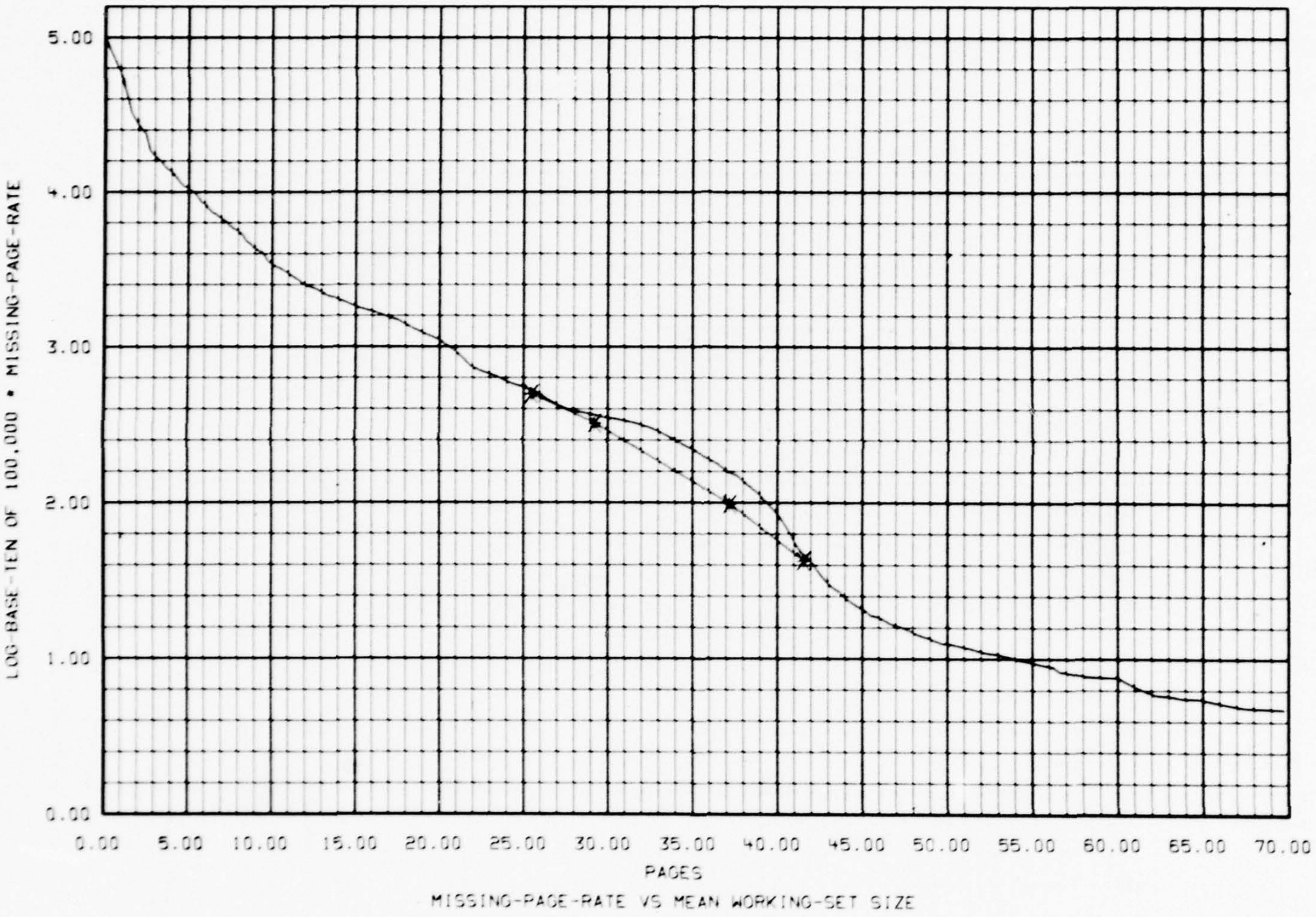


FIGURE 6-2 Comparison of OPTMEM to Pure WS for the First Trace of the MUDDLE Assembler

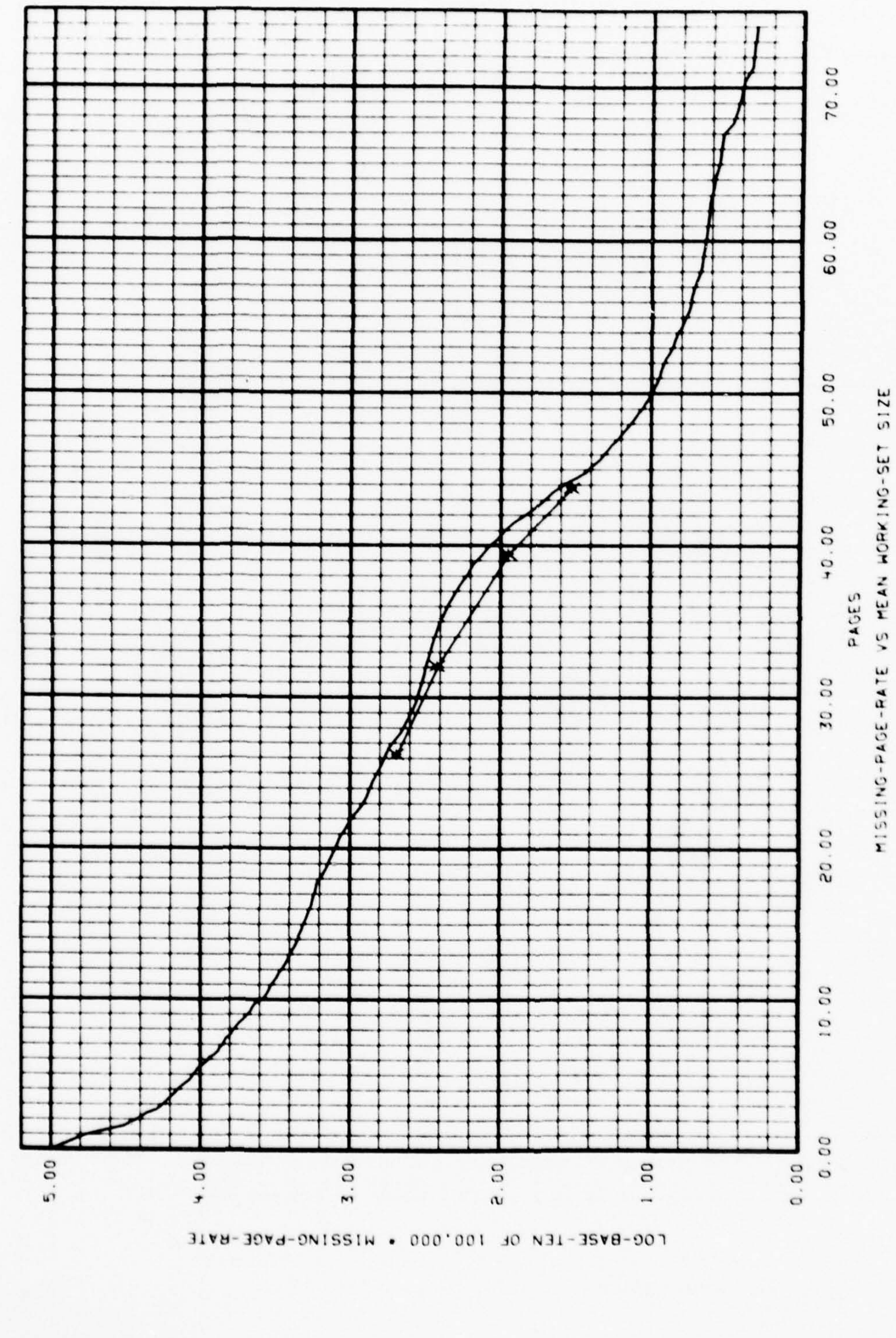


FIGURE 6-3 Comparison of OPTMEM to Pure WS for the Second Trace of the MUDDLE Assembler

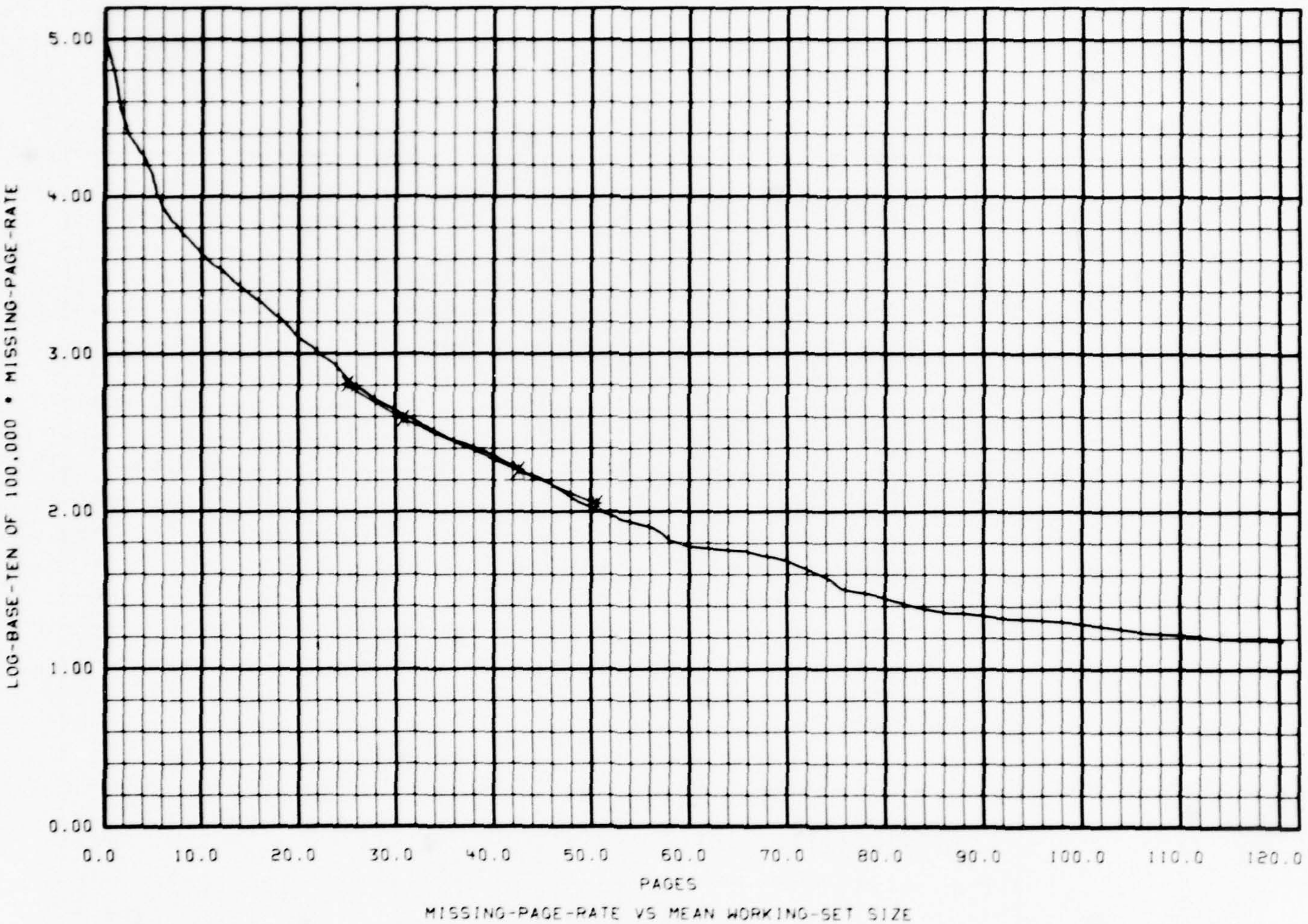


FIGURE 6-4 Comparison of OPTMEM to Pure WS for the MIDAS Assembler (First Trace)

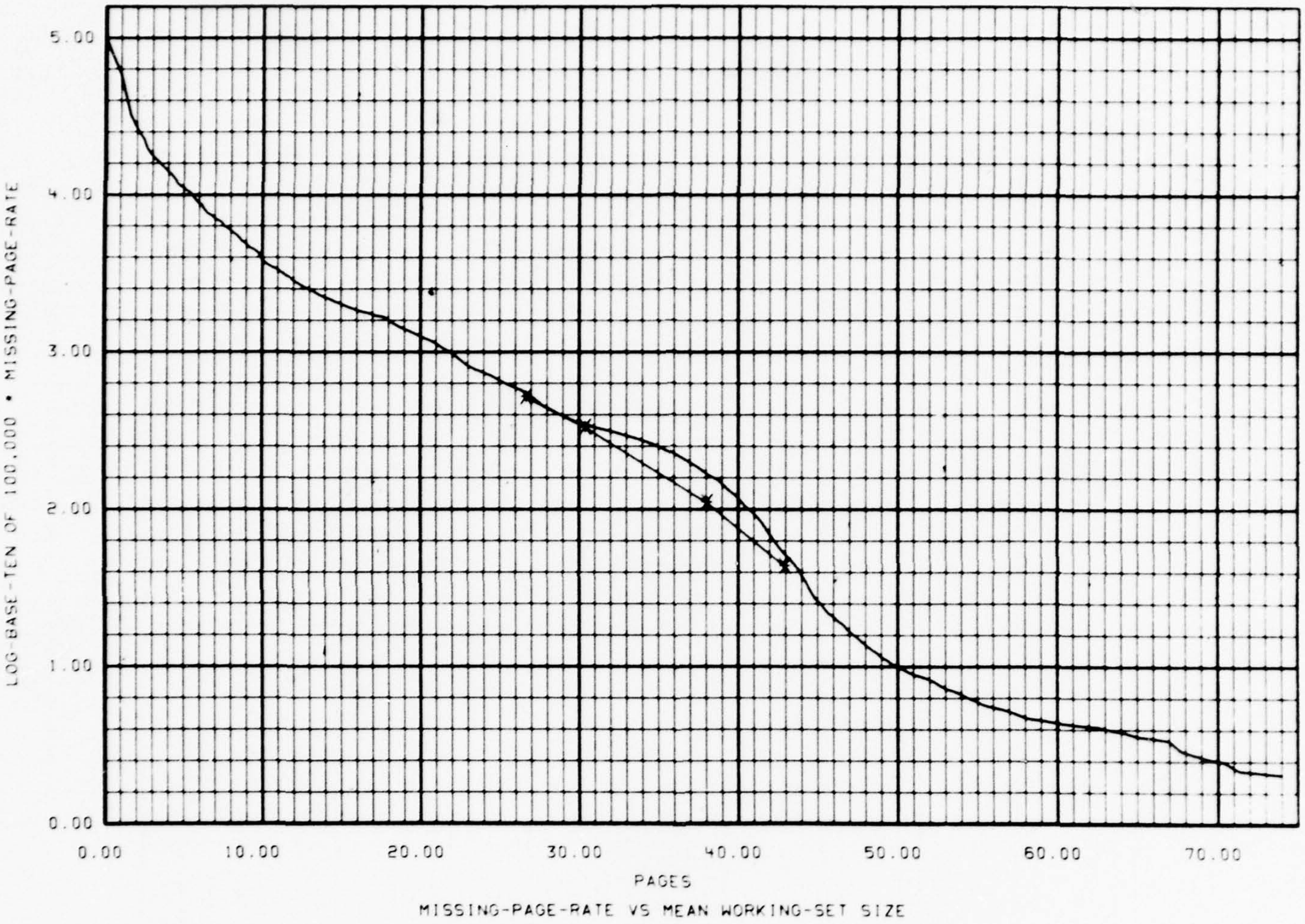


FIGURE 6-5 Results for First Trace's Predictor Used on the Second Trace of the MUDDLE Assembler

6.5 Conclusions

Guided by control-theoretic considerations, we have developed an optimal memory management scheme whose performance is equivalent or better than that of a working-set scheme when compared upon a fault-rate versus mean memory partition size basis. Moreover, there is no reasonable basis upon which one can expect to do better than for our optimal predictor-controller scheme. To do so would require modification of our program model or our performance criterion. To change our model of program behavior, one would have to demonstrate that other programs have a more complex behavior than we have observed in our Chapter 4 results. While our choice of minimal total mean square error (6.10) as a performance criterion may be disputed, the adjustment of the threshold value in our controller (6.42) allows one to trade-off the costs between the two types of errors our predictor can make.

CHAPTER 7

DIRECTIONS FOR FUTURE RESEARCH

7.1 Some Open Issues

The research reported in this thesis raises several issues which should be addressed. The first issue is whether our empirical trace data,which was obtained on the Dynamic Modeling System of MIT's Project MAC, is truly indicative of the behavior of all programs. One might question whether our data and consequently our results, are specific to the programming and run-time environments of the MIT system. Other investigators should be encouraged to obtain similar data and analysis for different operating systems.

A second issue which could easily be addressed to our data is whether there is a significant difference in the address referencing behavior between the instruction (pure) and the data (impure) segments of a program. Numerous other operating systems provide a dichotomy between instructions and data and consequently could provide data to address this issue.

A final issue relates to our use of average statistics (e.g., the average page autocovariance) which were fully justified in terms of our performance criterion. The open question is, however, what is the difference between individual page behaviors. If they are shown to have a high degree of variability, one might seek other performance measures for the memory subsystem.

7.2 Future Extensions

There are many directions along which the research reported in this thesis can be extended. The first and most important area should be to expand the analysis and modeling effort to include the full spectrum of resources in an operating system and their interactions. Specifically, one should determine cross-covariance functions between all system resource utilizations levels as well as their individual autocovariances. For those resources which show a high degree of interaction, the more inclusive models will provide better control policies. For other resources which show little cross coupling (cross-covariance), the decomposability models of Courtois [Cou75] will have been justified.

A longer term goal should be to get beyond the system-theoretic process view of an operating system and develop true input-output models for an operating system. To do so will require new designs or instrumentation of current systems to estimate demand parameters of the individual jobs as well as the system's state in response to them.

Another fruitful area for future investigation is to continue the analogy between our thresholded memory controller of Section 6.3 and the detection theory problem of communications theory. The operating system equivalent to the receiver operating characteristic (ROC) curves [Sel65] should provide an operating system design tool.

APPENDIX A

COST FUNCTIONS FOR OPERATING SYSTEMS

A.1 Introduction

This Appendix contains three principal sections. The first contains a discussion of the Brownian motion model as a stochastic model for the memory state component of an operating system. The second section is devoted to the derivation of a result (eqn. 1.17) for the mean first passage time to an absorbing barrier for such a process. The final section using this result and others from Chapter 1, presents a parametric set of level cost contours for a two resource system.

A.2 Brownian Motion Model

In a basic paper, Coffman and Ryan [CoR72] introduce a stationary Gaussian process model for the working-set size of a program. It follows that the memory state component $x_1(t)$ being the sum of the working-set sizes of the resident programs will also be a similar type process. By Bryant's [Bry75] and our (Appendix B) results, the validity of the Gaussian assumption for single program memory's requirements must be rejected. However, as Coffman and Ryan rightly point out, on a system wide basis by virtue of the central limit theorem, the memory state component should have a Gaussian distribution.

Even though Coffman and Ryan introduce the Gaussian process model, they make no use of it as a dynamical model. In their consideration of the partitioning of main memory, they take a "snapshot" approach which converts the random process model into a random variable. Their results thus reflect only the static probabilities related to the individual or to the sum of Gaussian random variables. Coffman and Ryan also introduce the more general Ornstein-Uhlenbeck process model but again make no use of it. For their approach to the memory partition problem, both processes will yield the same results in that they both have the same first order probability distribution.

More recently, Ryan and Coffman [RyC74] have introduced an immigration-death process as a *dynamic model* for program demands upon main memory. We, however, concur with the earlier Gaussian process model which, when specialized to one of independent increments is equivalent to a Brownian motion (also called the Wiener process) [Sch73]. The Brownian motion or Wiener process may also be regarded as the continuous limit of a simple random walk [CoM65].

A.3 First Passage Time for Brownian Motion

Thrashing [Den68c] is a phenomenon in operating systems often observed when the level of memory utilization approaches the limit of physical memory. Since thrashing is counter-productive, a reasonable question to ask of any dynamical model for memory utilization is how frequent is physical memory exceeded. Even better, one may ask whatis the mean time between starting from some specified level of usage till when physical memory is exceeded. The answer is provided by the solution of the so-called "first passage time" or "ruin" problems for various stochastic models. Such problems require the specification of boundary conditions which are usually given such descriptive designations as absorbing, elastic, or reflecting barriers.

Here we are specifically interested in the first passage time for our Brownian motion model of memory utilization level $X(t) = x_1(t)$ to an absorbing barrier at $x = b$ representing the limit of physical memory. For computer operating systems, there is a problem with the appropriate boundary condition at the low end where the system goes empty. The problem at $x=0$ is twofold. First, there is the problem of physical interpretation and secondly, there is the larger problem of existence and uniqueness of the solution of the resulting mathematical problem. Our choice is on the side of mathematical tractability and we thus settle upon a reflecting barrier as the boundary condition of $x = 0$. A pursuit of the other aspect of the problem leads to singular diffusion equations which excludes any boundary condition at $x = 0$ [B-R60].

In Chapter 1, the solution was given for the mean first passage time (to absorption) for a Brownian Motion or Wiener process with drift $\mu = 0$ and variance parameter σ^2 starting from $X(0) = x_0$ and subject to a reflecting barrier at $x = 0$ and an absorbing barrier at $x = b$. Here we develop the more general (μ not zero) solution following the proceedure given in Chapter 5 of Cox and Miller [CoM65].

Specifically, we view the probability distribution of the current memory state $X(t)$ as a probability density.

$$u(x,\ t) = \text{Prob. density of } X(t) \text{ at } x, \tag{A.1}$$

which one can interpret as follows:

$$u(x,\ t)\,dx = \text{Prob } [x < X(t) < x + dx] \tag{A.2}$$

Then it is well known that u satisfies a pair of diffusion (Kolmogorov) equations. Since our object is to determine the first passage time distribution as a function of initial state x_0, it is the backward equation which is appropriate, namely

$$\frac{1}{2}\sigma^2 \frac{\partial^2 u}{\partial x_0^2} + \mu \frac{\partial u}{\partial x_0} = \frac{\partial u}{\partial t}\,. \tag{A.3}$$

Let T be the required first passage time and $g(t|x_0)$ be its probability density function with

$$1 - G(t) = \text{Prob}[T > t] = \int_0^b u(x,\ t;\ x_0)dt \tag{A.4}$$

where $G'(t) = g(t|x_0)$. Since $g(t|x_0) = -\frac{\partial}{\partial t} u(x,\ t;\ x_0)$, taking Laplace transforms, we have

$$\gamma(x_0) = g^*(s|x_0) = \int_0^\infty e^{-st}\, g(t|x_0)dt = -su^*(x,\ s;\ x_0) \tag{A.5}$$

Also, transforming the backward equation (A.3), we require

$$\frac{1}{2}\sigma^2 \frac{d^2\gamma}{dx_0^2} + \mu \frac{d\gamma}{dx_0} = -s\,\gamma\,(x_0) \tag{A.6}$$

which we must solve subject to the boundary conditions (see [CoM65], p. 231)

$$\gamma'(0) = 0, \qquad \gamma(b) = 1. \tag{A.7}$$

Letting

$$\theta_1(s) = \frac{1}{\sigma^2}\left(-\mu - \sqrt{\mu^2+2\sigma^2 s}\right), \quad \theta_2(s) = \frac{1}{\sigma^2}\left(-\mu + \sqrt{\mu^2+2\sigma^2 s}\right) \tag{A.8}$$

be the roots of the characteristic equation (A.6), we can write the general solution of (A.6) as

$$\gamma(x_0) = A\,\exp(x_0\theta_1(s)) + B\,\exp(x_0\theta_2(s)). \tag{A.9}$$

Applying boundary conditions (A.7),

$$\gamma(x_0) = \frac{\theta_2 e^{x_0\theta_1} - \theta_1 e^{x_0\theta_2}}{\theta_2 e^{b\theta_1} - \theta_1 e^{b\theta_2}} \tag{A.10}$$

When $\mu = 0$, the above specializes to

$$\gamma(x_0) = \frac{\cosh(x_0\theta(s))}{\cosh(b\theta(s))} \quad . \tag{A.11}$$

In either case, using

$$\bar{T} = -\left.\frac{d\gamma}{ds}\right|_{s=0} \tag{A.12}$$

we find for $\mu \neq 0$

$$\overline{T} = \frac{1}{\mu}(b - x_o) + \frac{\sigma^2}{2\mu^2}\left\{\exp(-\frac{2\mu b}{\sigma^2}) - \exp(-\frac{2\mu x_o}{\sigma^2})\right\} \tag{A.13}$$

and for $\mu = 0$

$$\overline{T} = \frac{b^2 - x_o^2}{\sigma^2} \tag{A.14}$$

which is the result given in Chapter 1 as equation (1.17).

A.4 Operating System (Level) Cost Contours

Using the reciprocal of the mean time to absorption as a rate for how often physical memory is exceeded (with its subsequent overhead), one can model the memory cost by a function of the form

$$C_1(x_1) = (1 - x_1) + \beta_{1k} \frac{\sigma^2}{1 - x_1^2} , \tag{A.15}$$

where the coefficient β_{1k} reflects some subjective value judgement between the cost of more hardware verses more system overhead. With CPU costs derived from a simple queueing model as in Chapter 1, cf., equation (1.15)

$$C_2(x_2) = (1 - x_2) + \beta_{2k} \frac{x_2(1 + C_\nu^2)}{2(1 - x_2)} , \tag{A.16}$$

we can combine the two individual costs into a (two resource) system cost function

$$C(\underset{\sim}{x}) = C(x_1, x_2) = \alpha_1 C_1(x_1) + \alpha_2 C_2(x_2). \tag{A.17}$$

On the pages following are a parametric set of level cost contours for the system function (A.17) for the subjective relative cost parameters,

$$\beta_{1k} = \beta_{2k} = 0.05k, \quad k = 1, 2, \ldots, 6. \tag{A.18}$$

Memory cost were fixed at twice those for the CPU (i.e., $\alpha_1 = 2\alpha_2$) and the other coefficients were fixed with

$$\sigma^2 = C_\gamma^2 = 1.0 \tag{A.19}$$

One can observe that as the overhead portion increases (i.e., β's increasing), the desired operating point (minimum) moves to lower utilization levels.

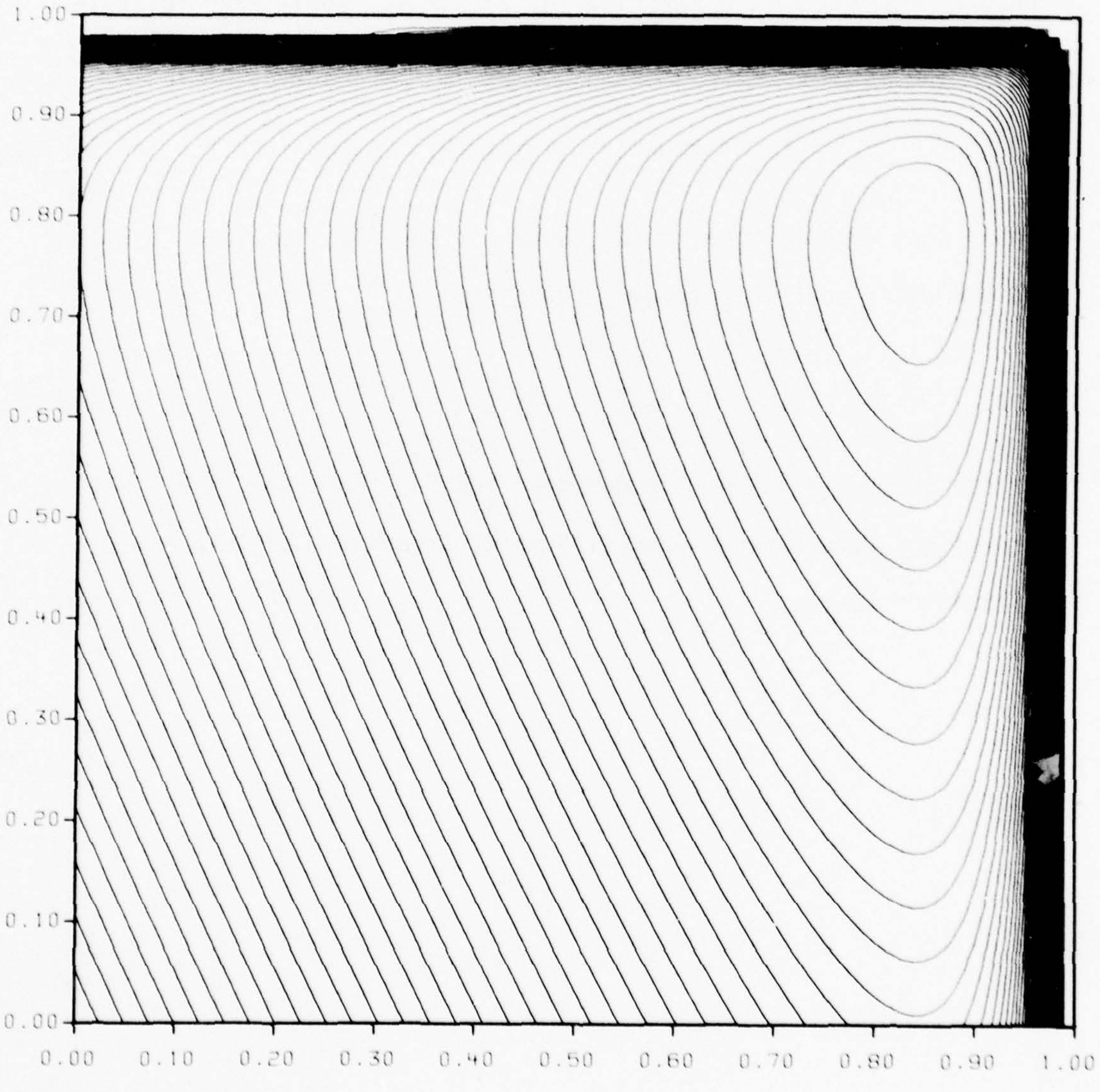


FIGURE A-1 Operating System Level Cost Contours (β's = 5%)

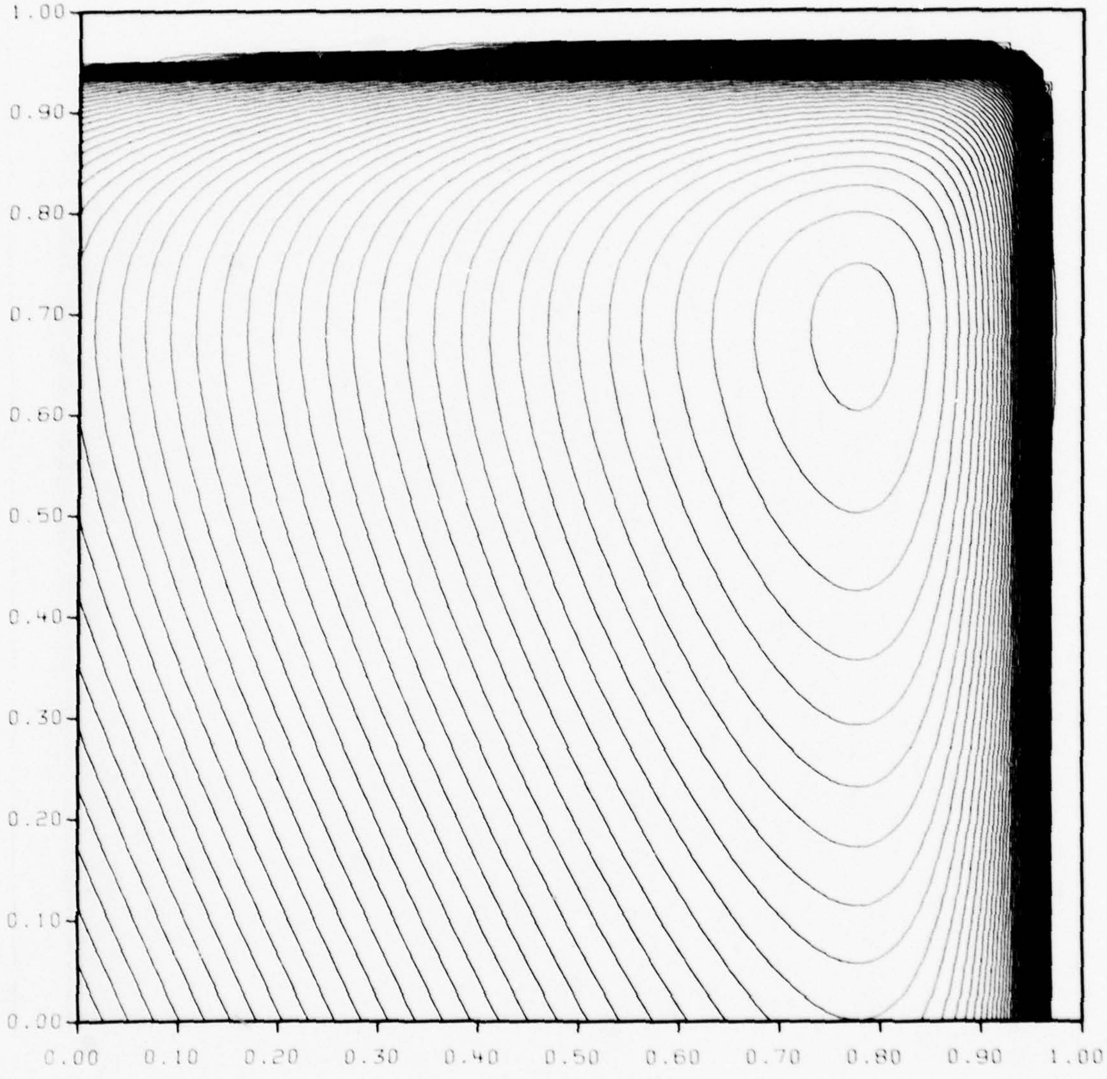


FIGURE A-2 Operating System Level Cost Contours (β's = 10%)

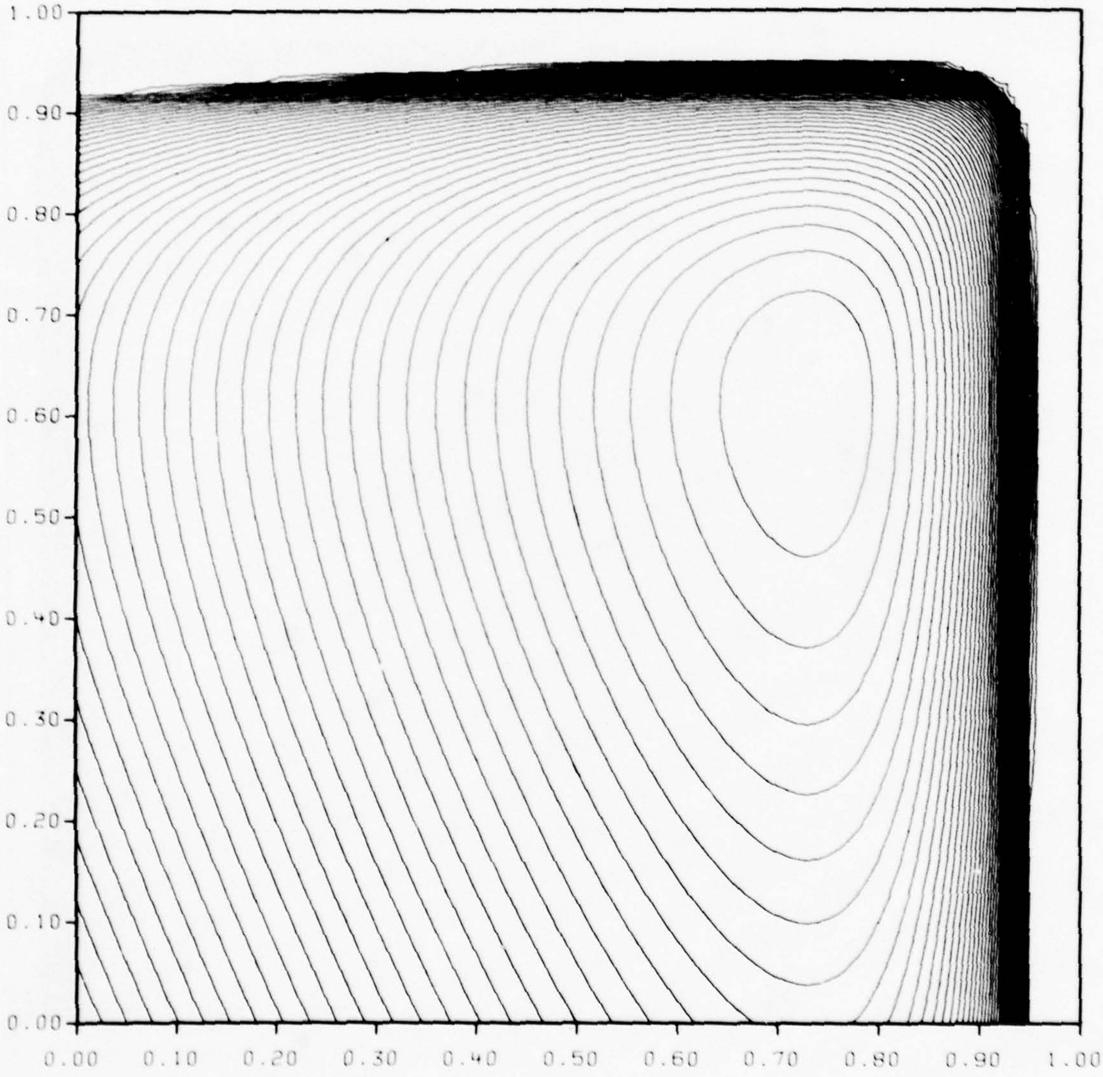


FIGURE A-3 Operating System Level Cost Contours (β's = 15%)

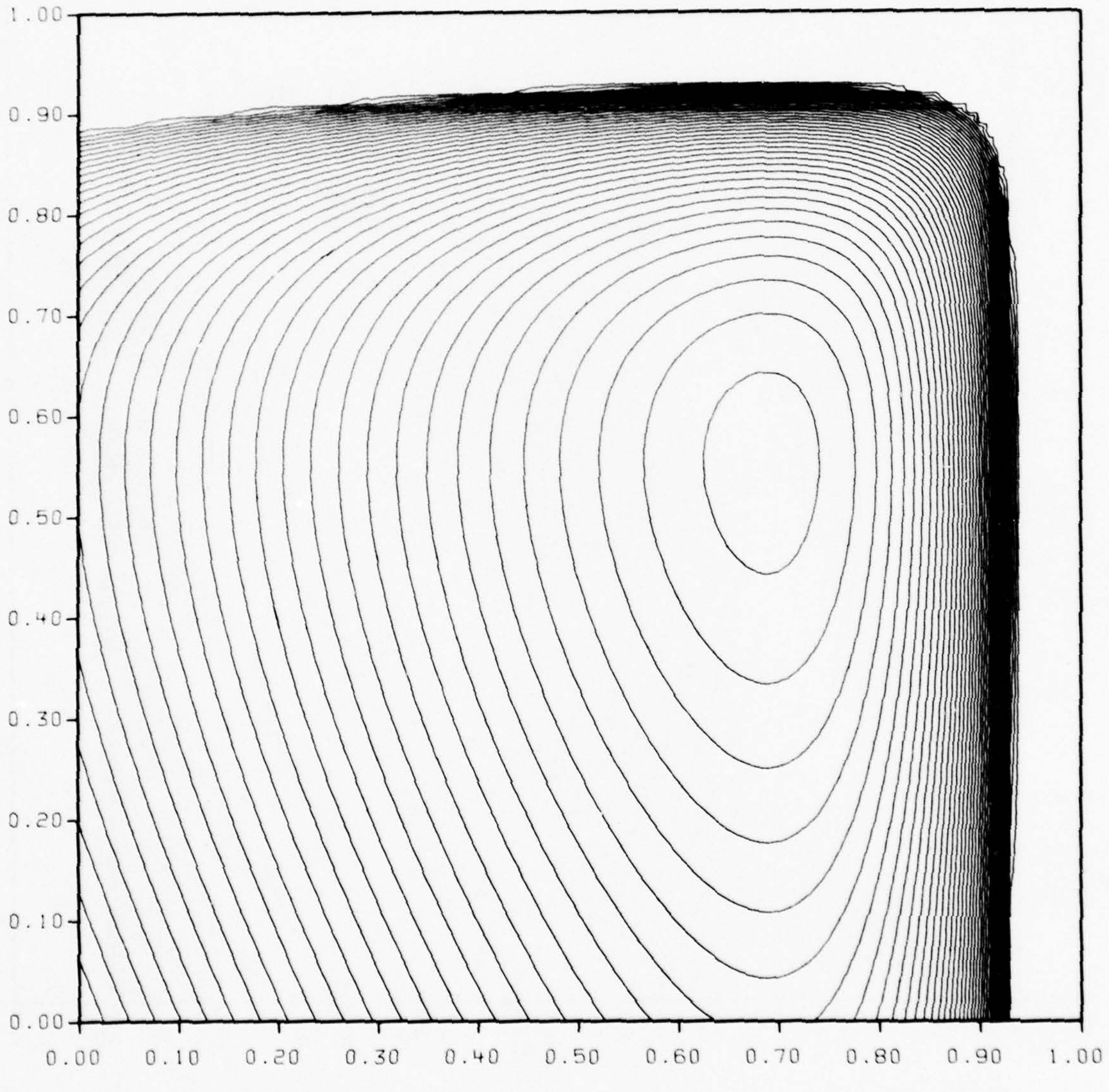


FIGURE A-4 Operating System Level Cost Contours (β's = 20%)

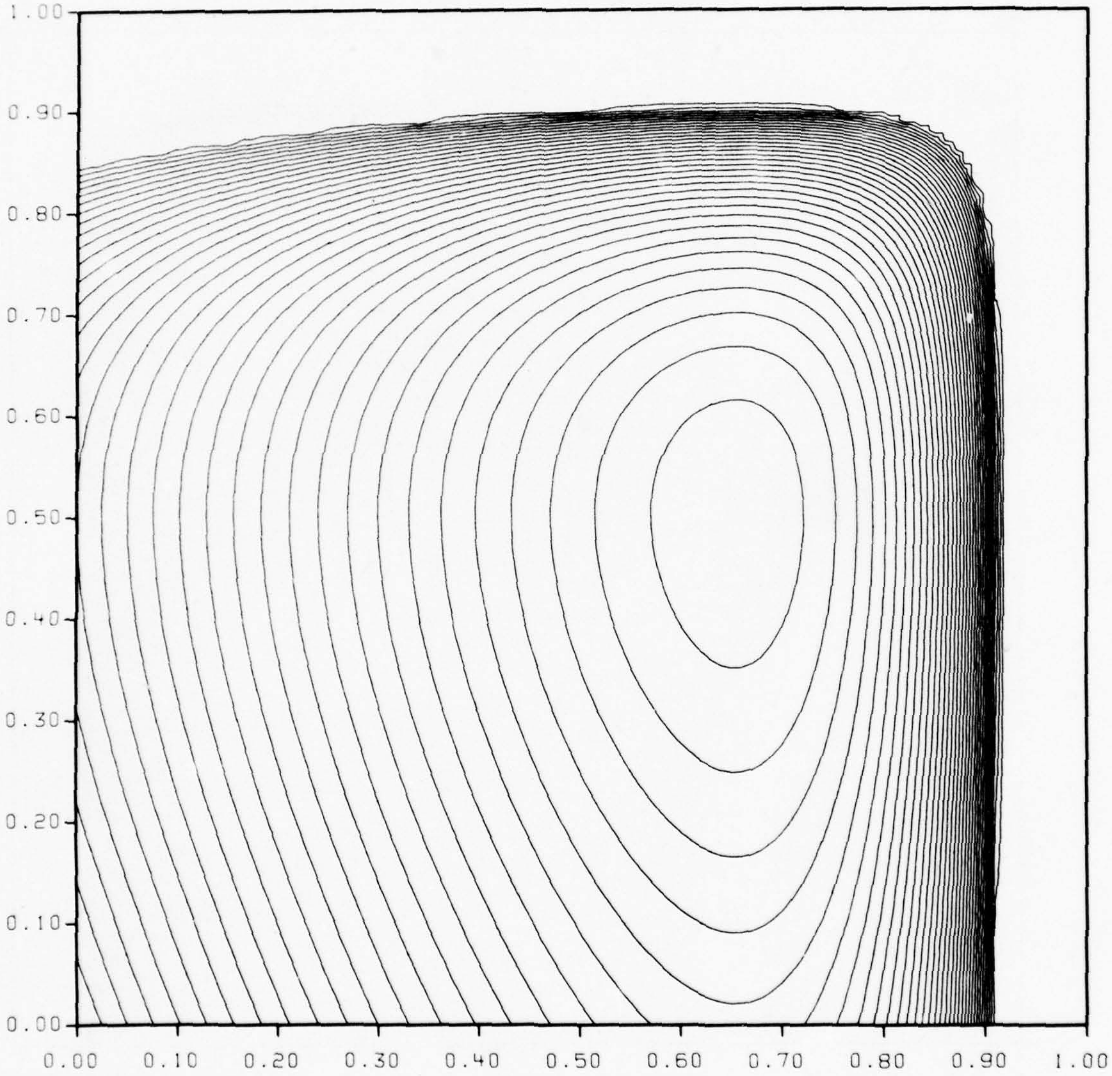


FIGURE A-5 Operating System Level Cost Contours (β's = 25%)

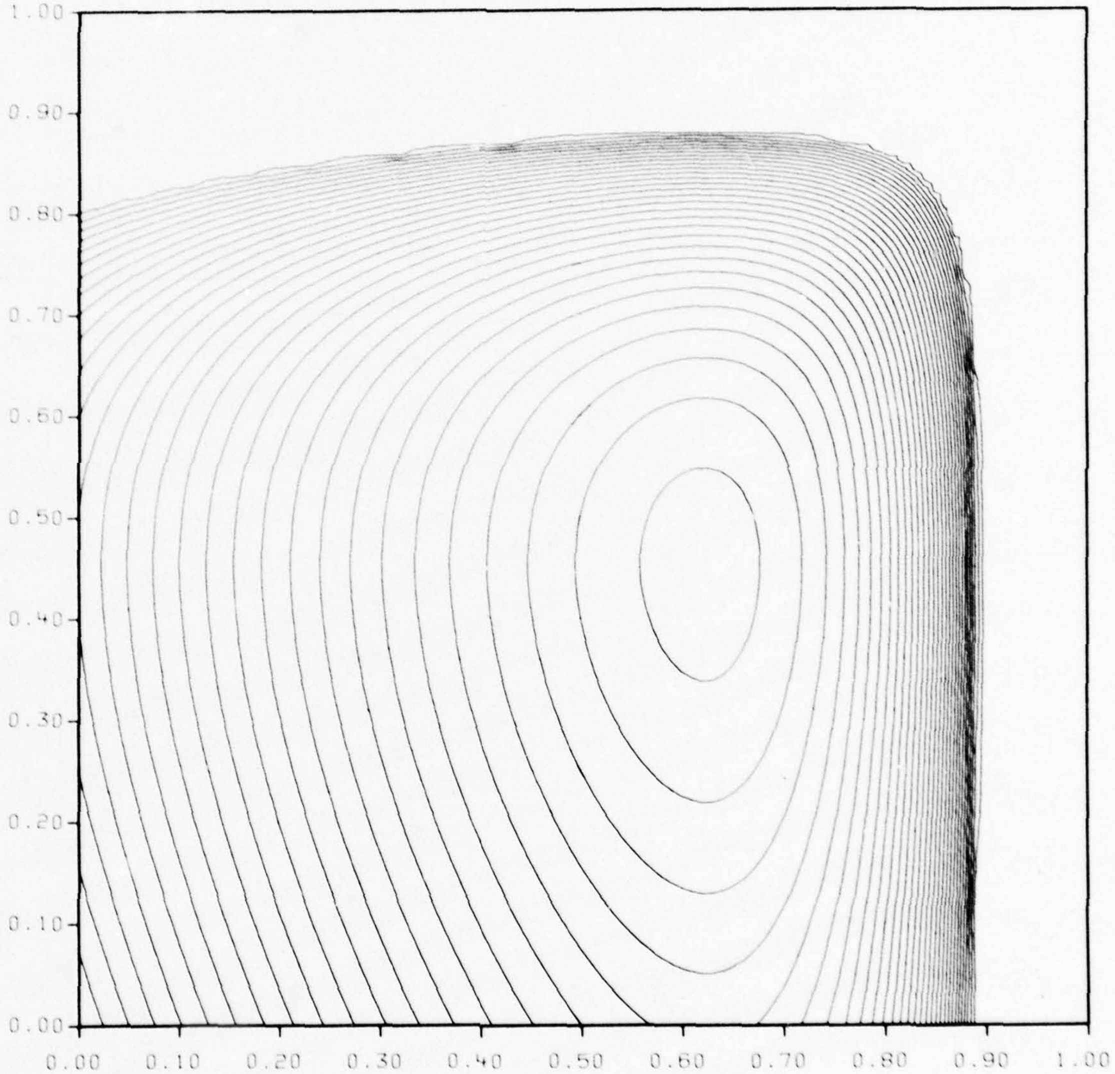


FIGURE A-6 Operating System Level Cost Contours (β's = 30%)

APPENDIX B

ADDITIONAL FIRST ORDER STATISTICS

OF

SAMPLE PROGRAM TRACES

This appendix contains a fairly complete listing of all of the first-order statistical results derived from the sample trace tapes as described in Chapter 3.

APPENDIX B

This appendix contains a fairly complete listing of all of the first-order statistical results derived from the sample trace tapes as described in Chapter 3.

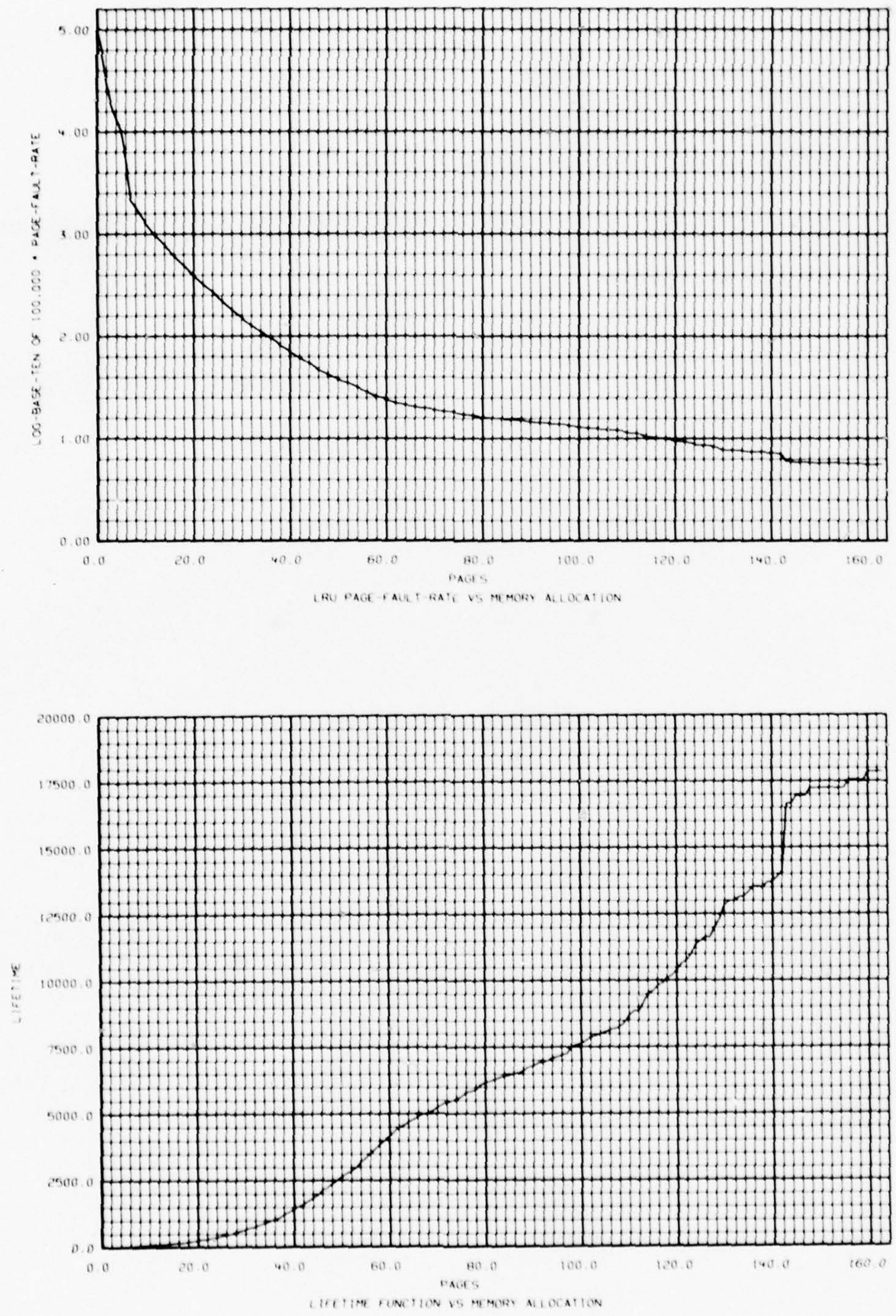
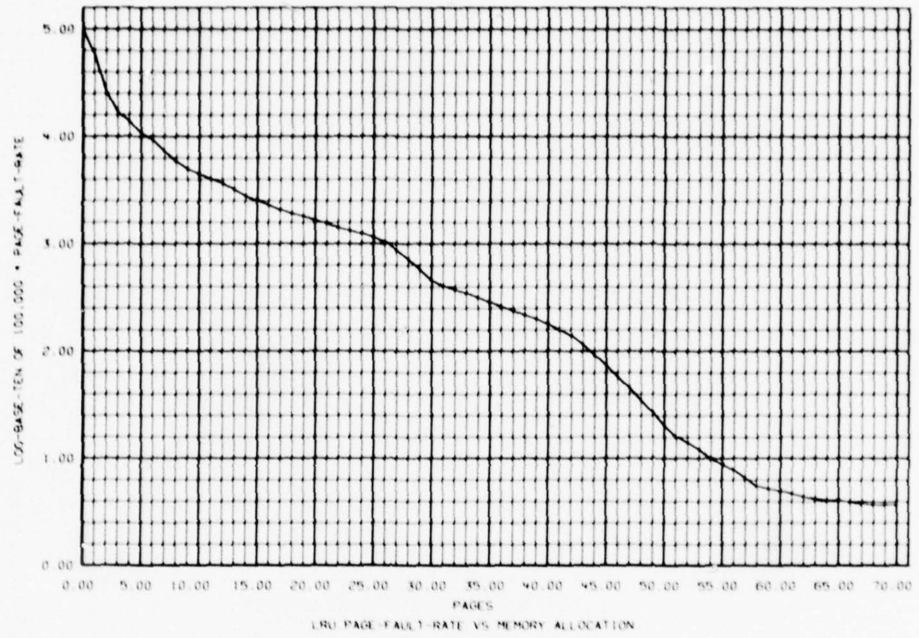


FIGURE B-1 LRU Analysis for the MUDDLE Compiler



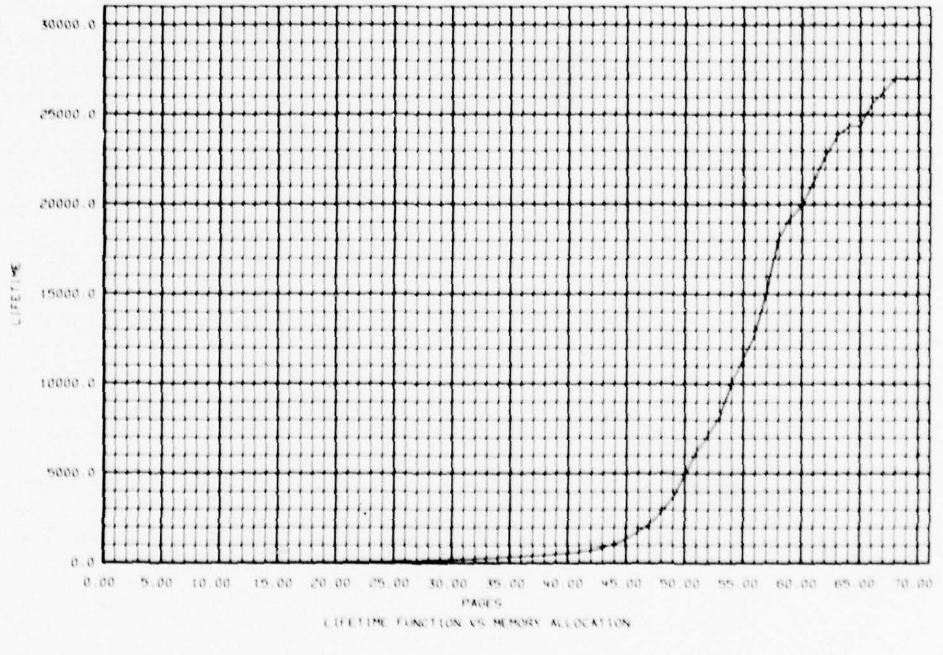


FIGURE B-2 LRU Analysis for the MUDDLE Assembler (First Trace)

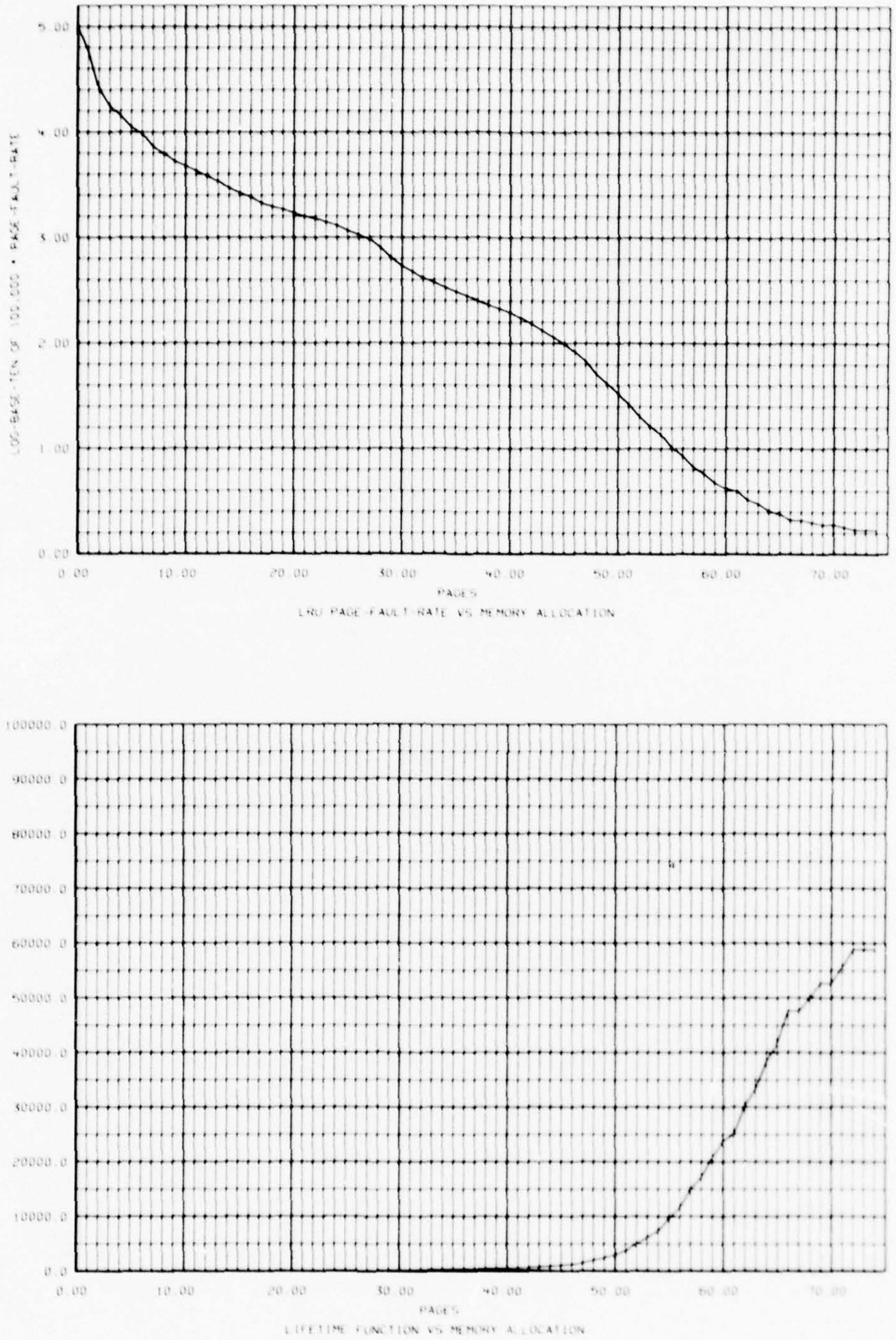


FIGURE B-3 LRU Analysis for the MUDDLE Assembler (Second Trace)

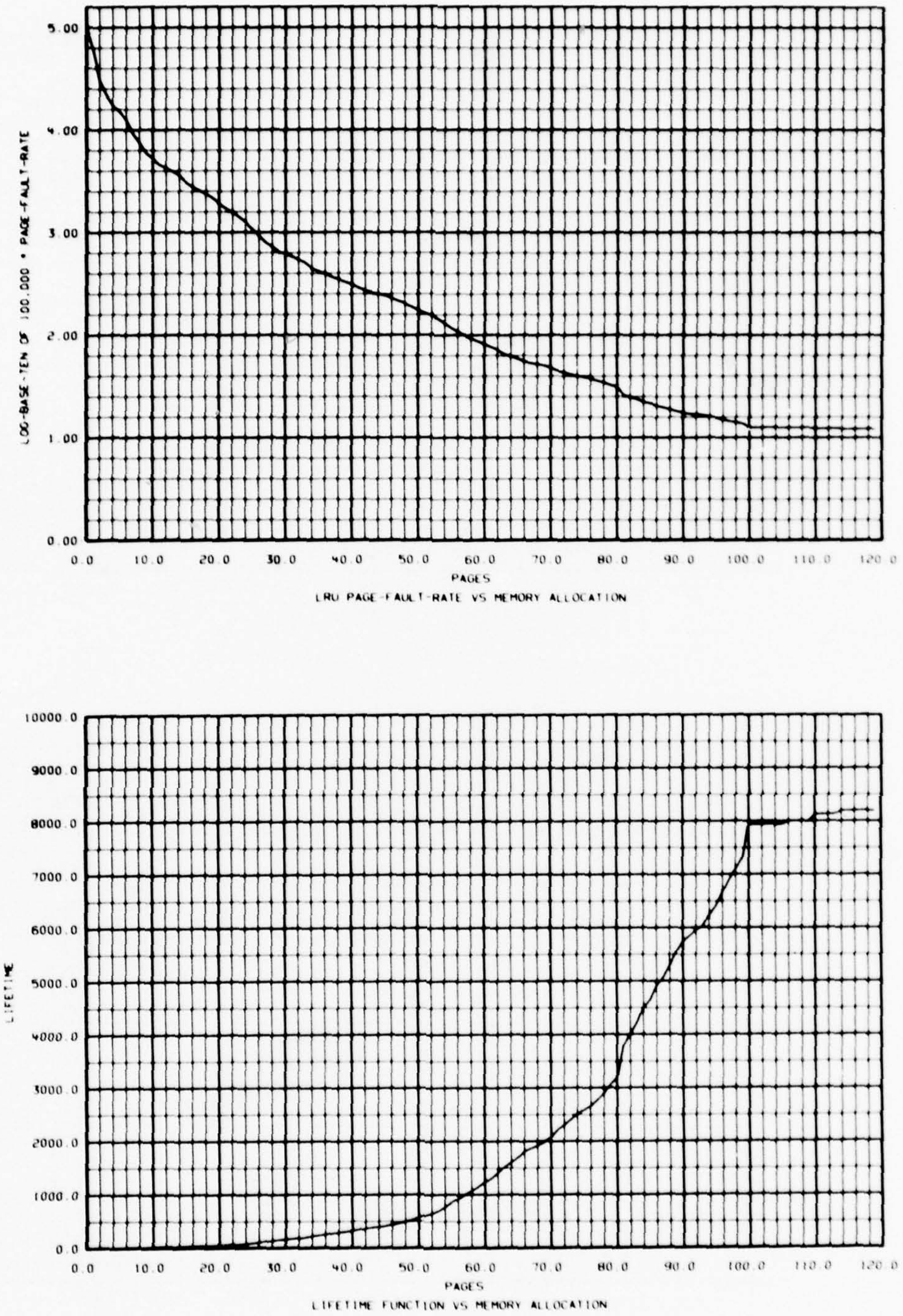


FIGURE B-4 LRU Analysis for the MIDAS Assembler (First Trace)

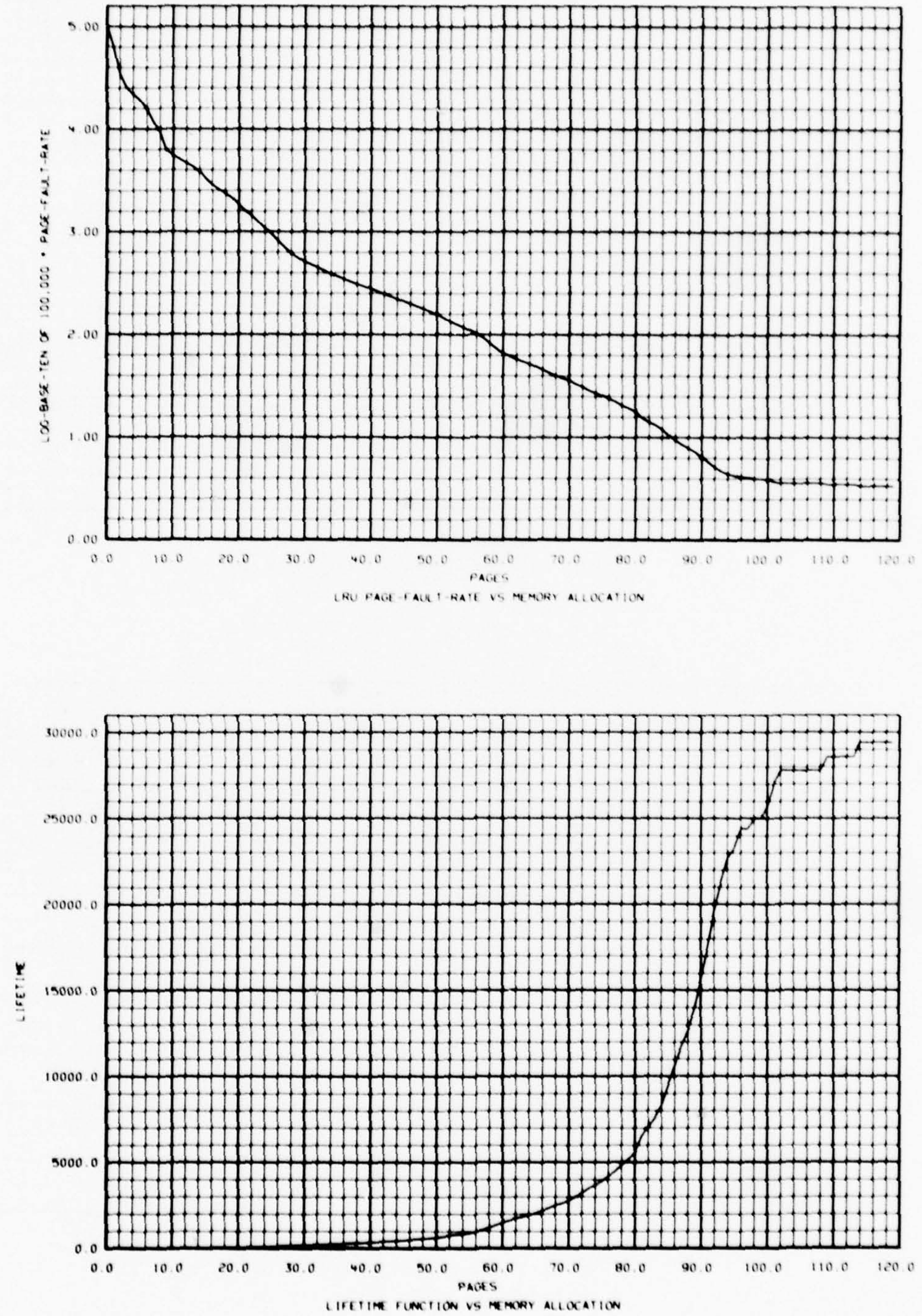
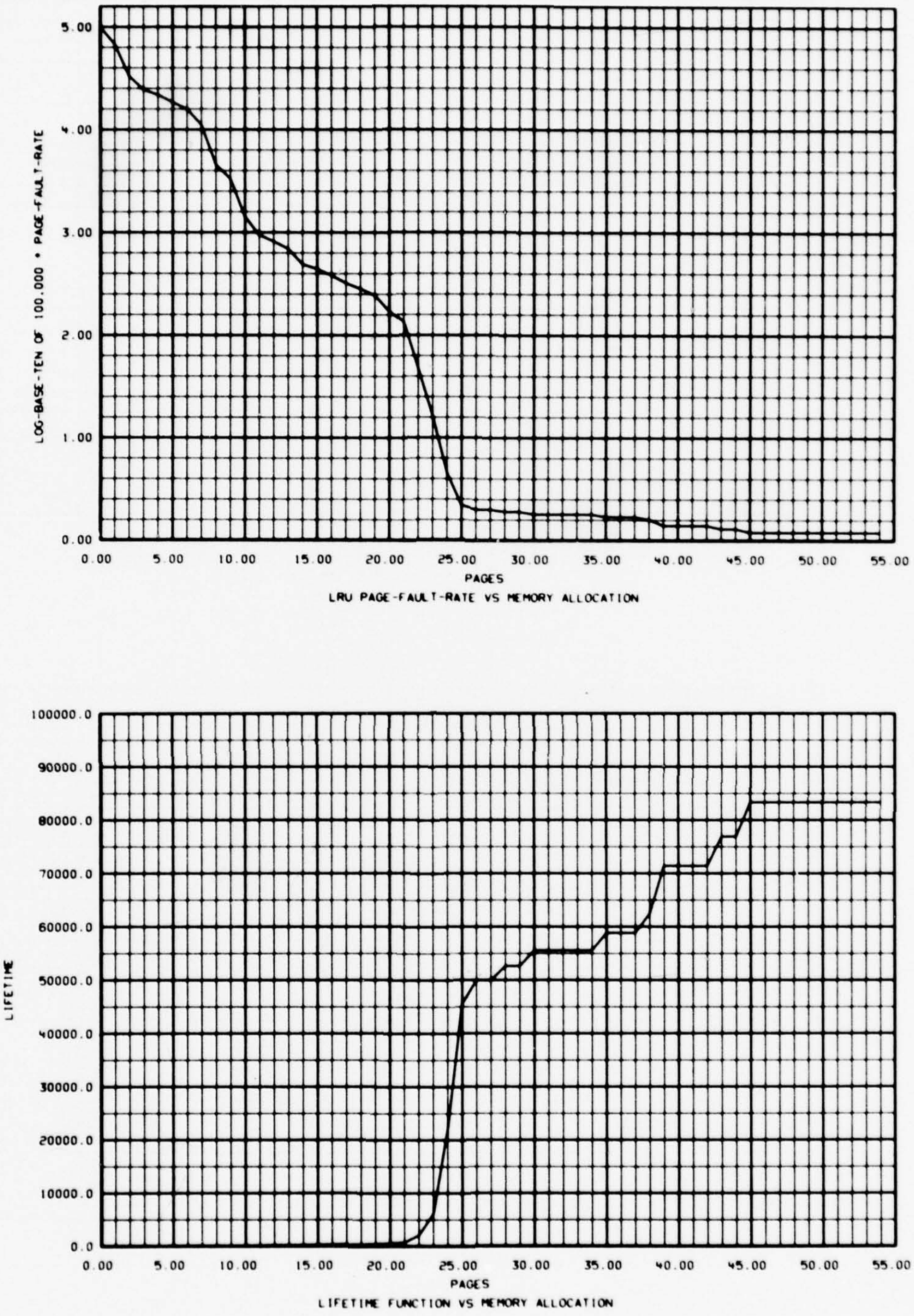


FIGURE B-5 LRU Analysis for the MIDAS Assembler (Second Trace)



LRU PAGE-FAULT-RATE VS MEMORY ALLOCATION

LIFETIME FUNCTION VS MEMORY ALLOCATION

FIGURE B-6 LRU Analysis for the Text Editor TECO

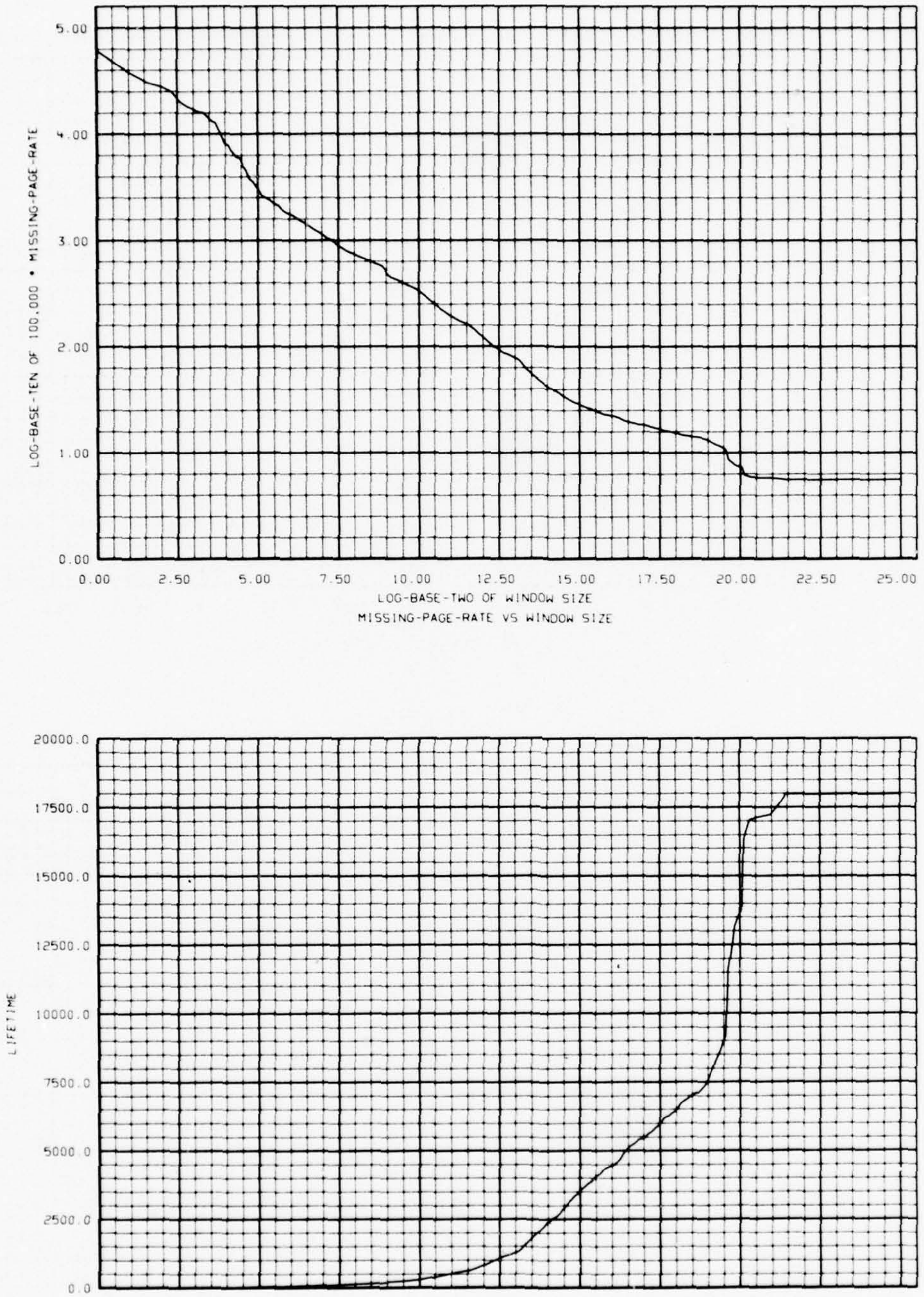


FIGURE B-7 Working-Set Analysis for the MUDDLE Compiler

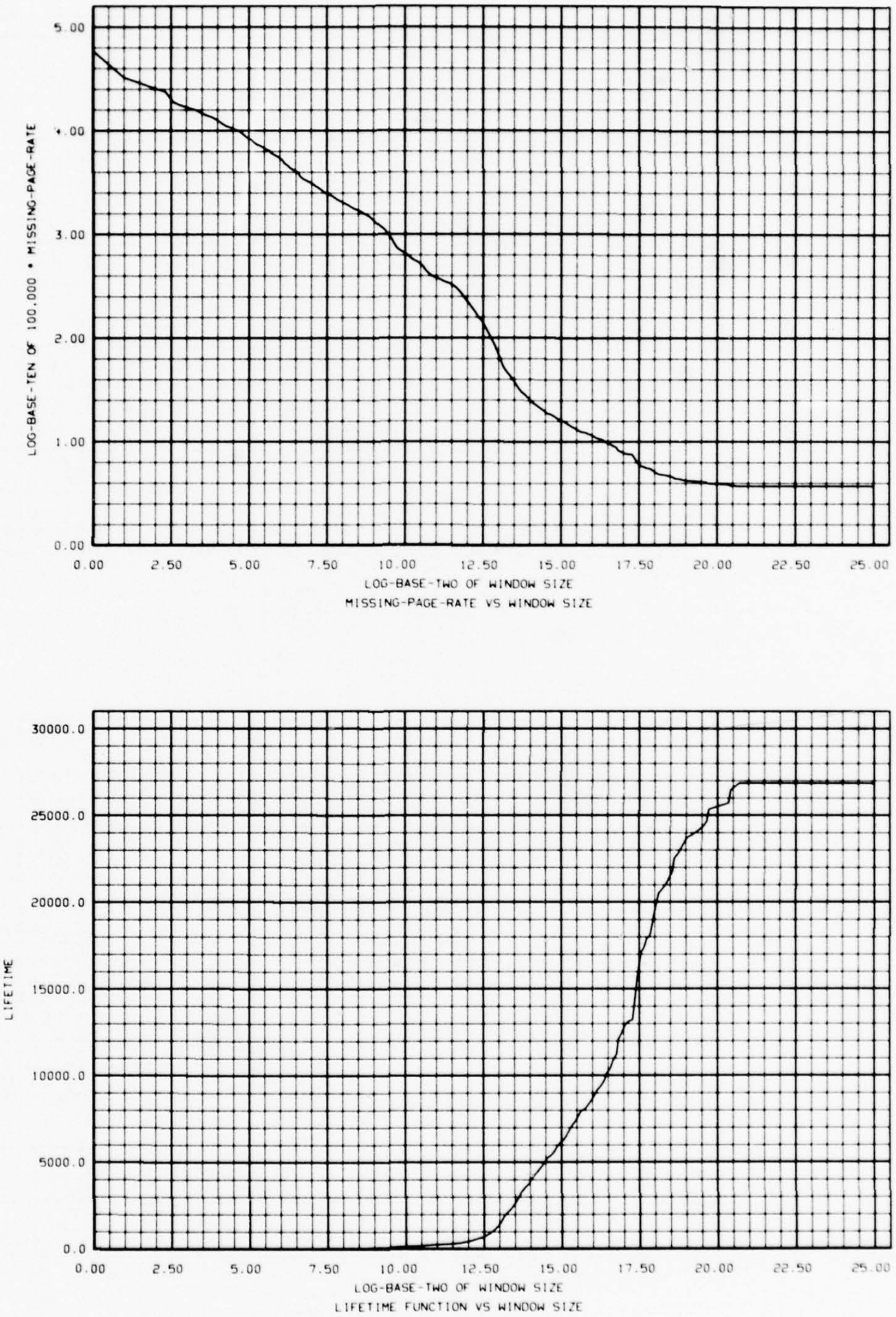


FIGURE B-8 Working-Set Analysis for the MUDDLE Assembler (First Trace)

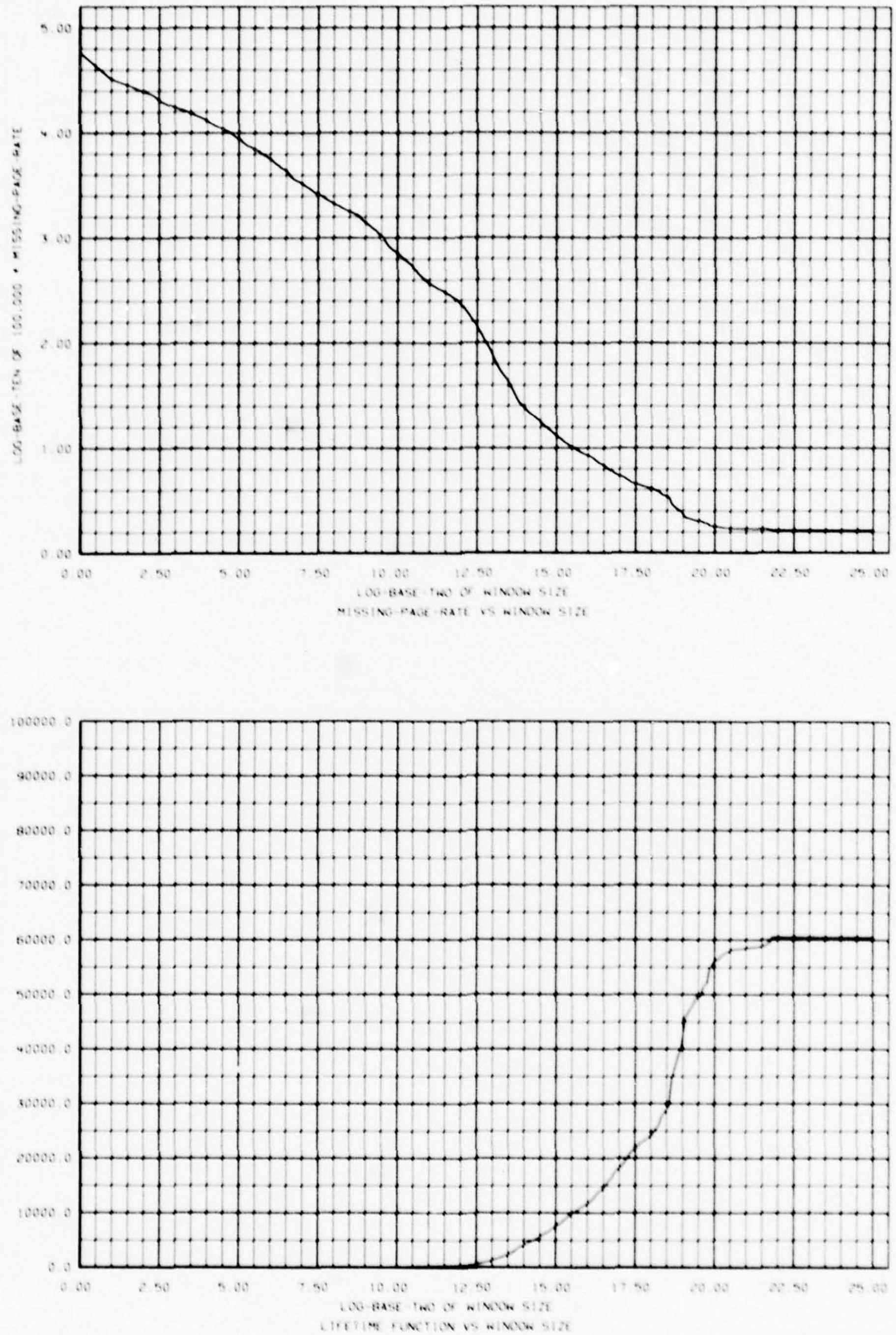
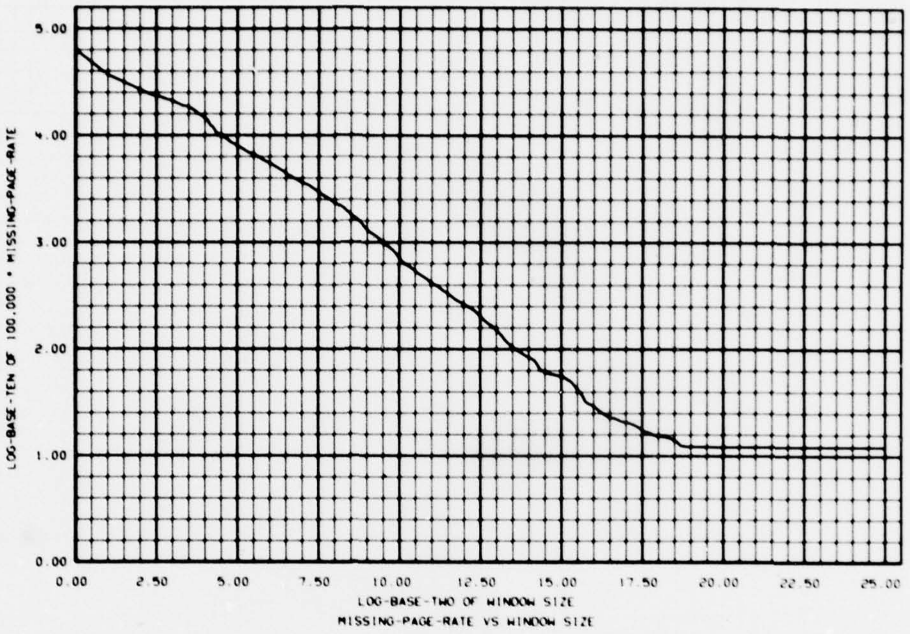


FIGURE B-9 Working-Set Analysis for the MUDDLE Assembler (Second Trace)



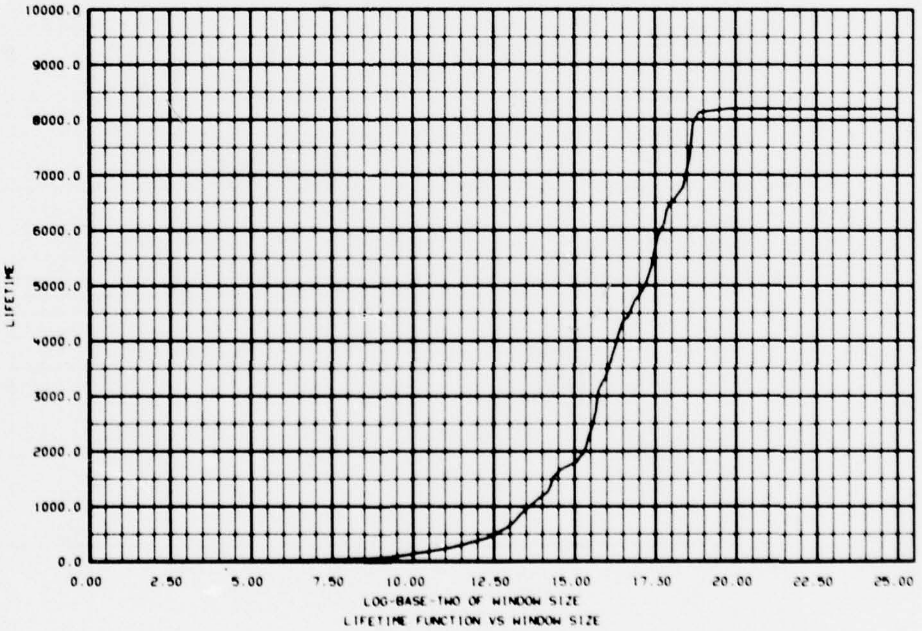


FIGURE B-10 Working-Set Analysis for the MIDAS Assembler (First Trace)

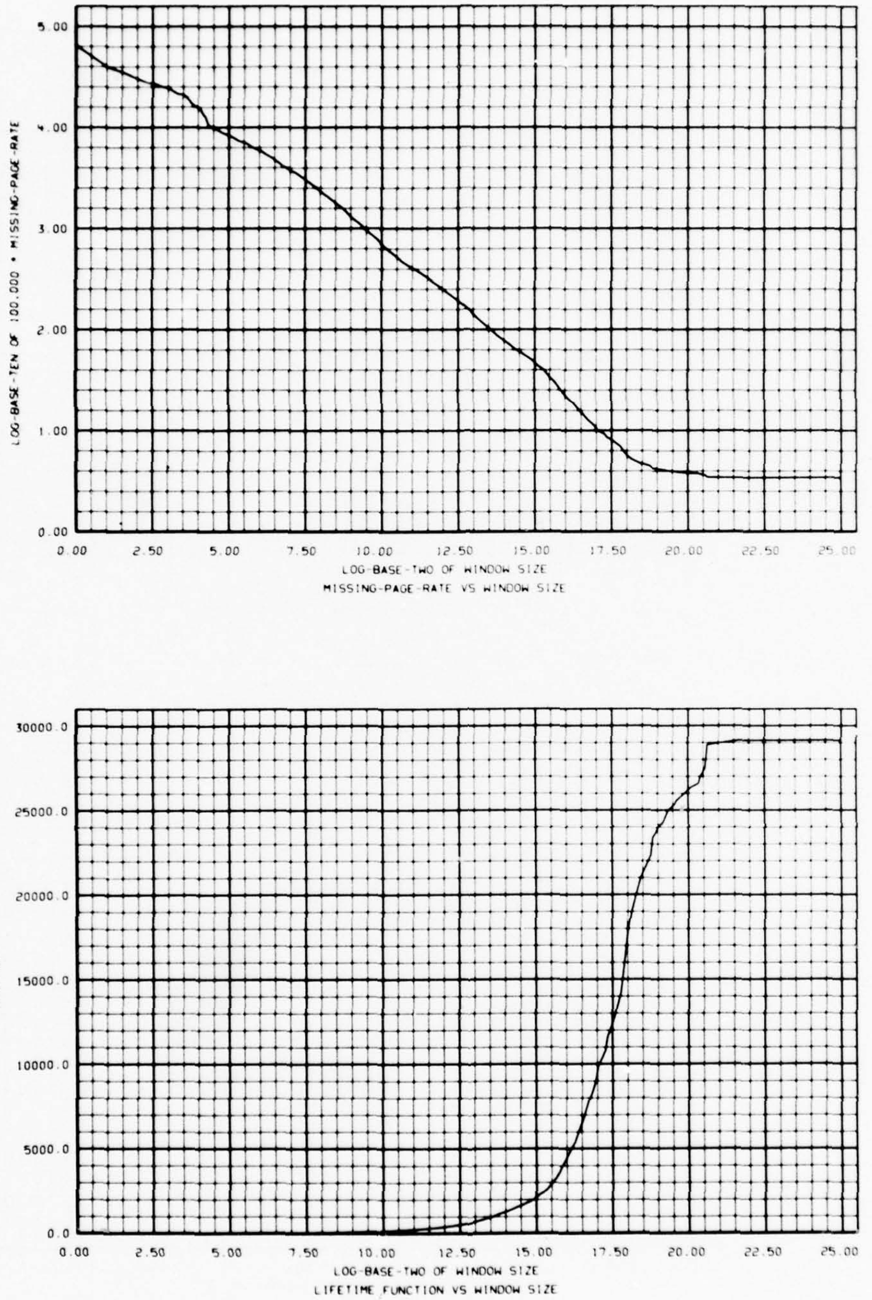
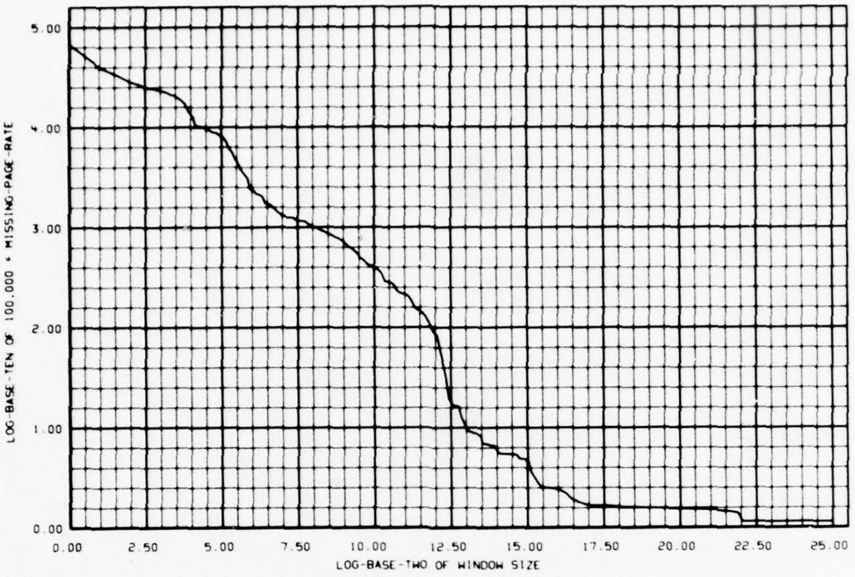
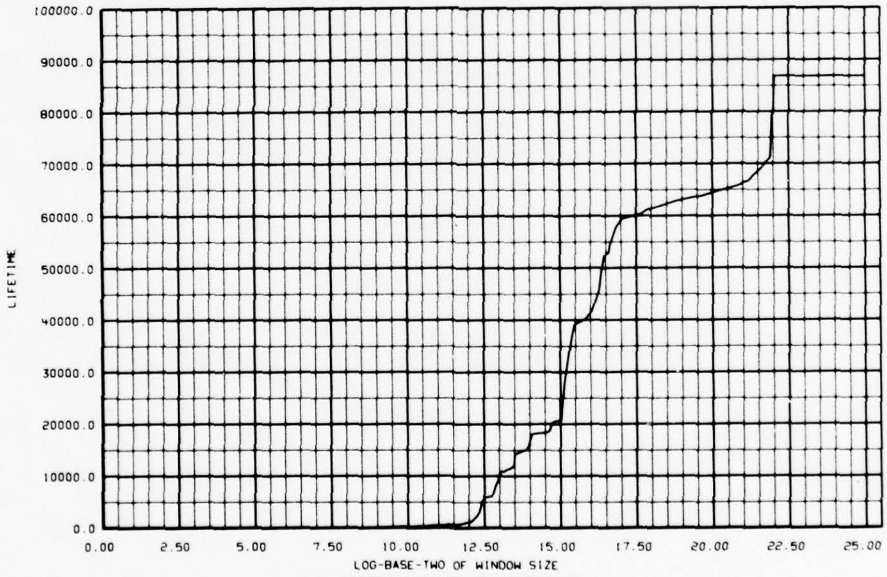


FIGURE B-11 Working-Set Analysis for the MIDAS Assembler (Second Trace)



MISSING-PAGE-RATE VS WINDOW SIZE



LIFETIME FUNCTION VS WINDOW SIZE

FIGURE B-12 Working-Set Analysis for the Text Editor TECO

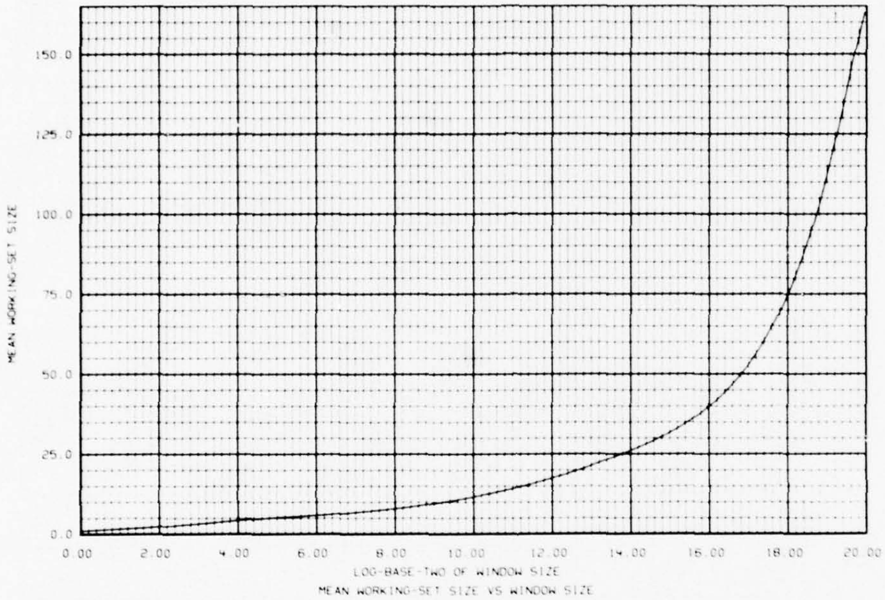


FIGURE B-13 Mean Working-Set Size vs. Window Size for the MUDDLE Compiler

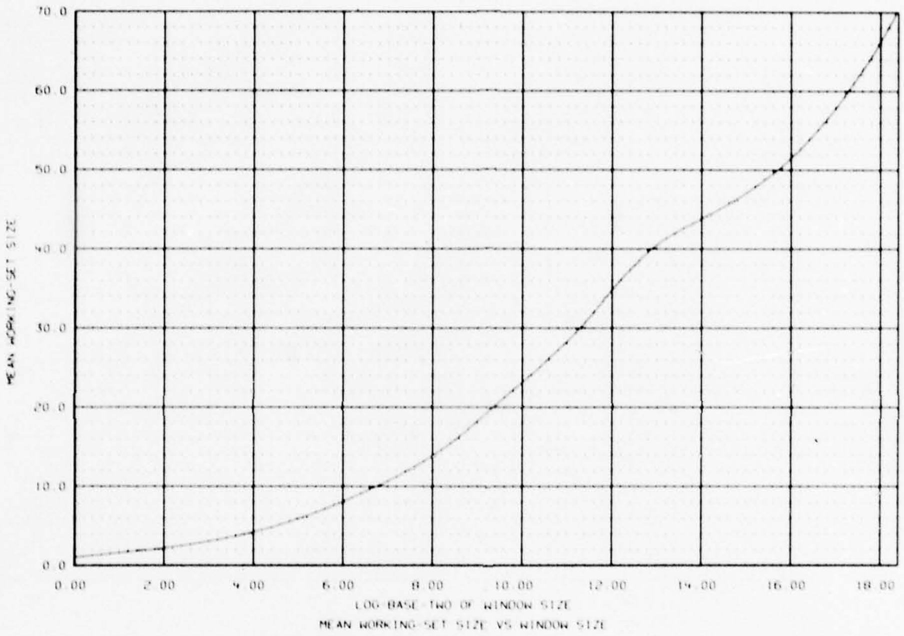


FIGURE B-14 Mean Working-Set Size vs. Window Size for the MUDDLE Assembler (First Trace)

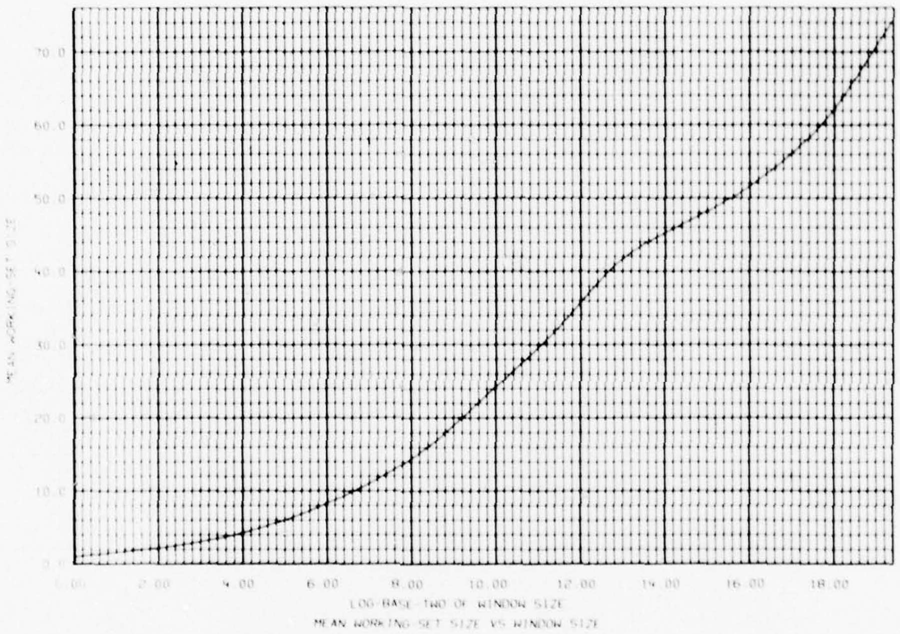


FIGURE B-15 Mean Working-Set Size vs. Window Size for the MUDDLE Assembler (Second Trace)

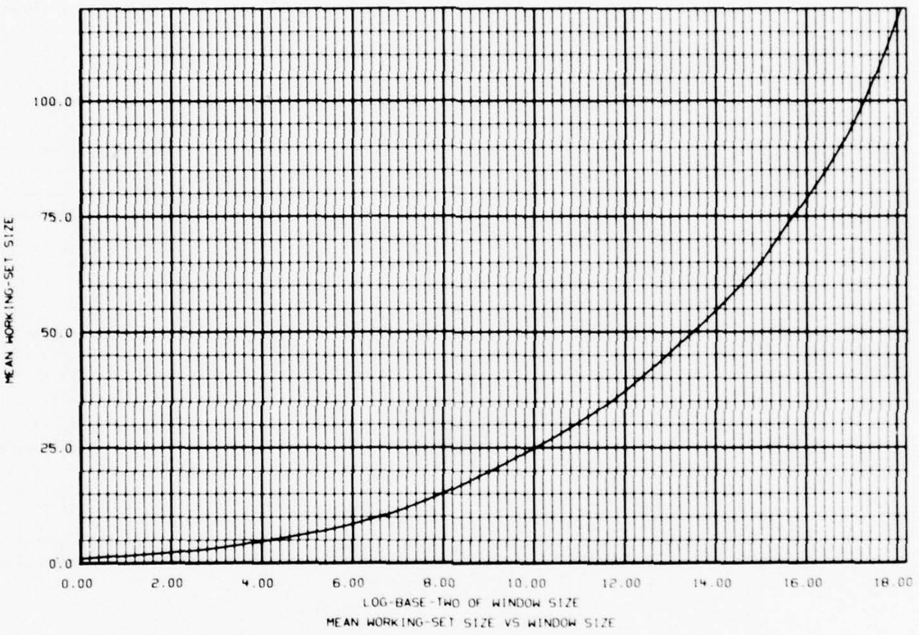


FIGURE B-16 Mean Working-Set Size vs. Window Size for the MIDAS Assembler (First Trace)

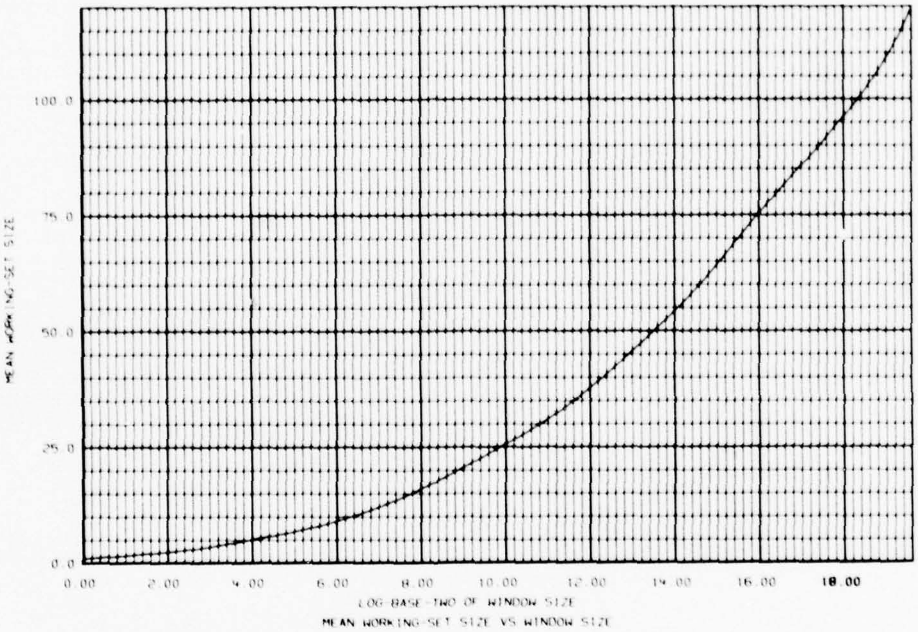


FIGURE B-17 Mean Working-Set Size vs. Window Size for the MIDAS Assembler (Second Trace)

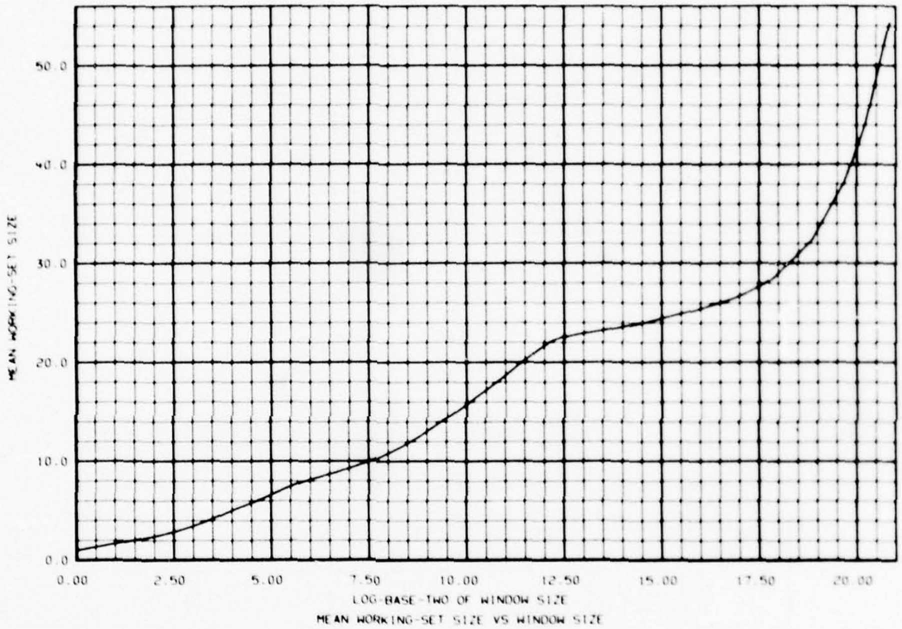


FIGURE B-18 Mean Working-Set Size vs. Window Size for the Text Editor TECO

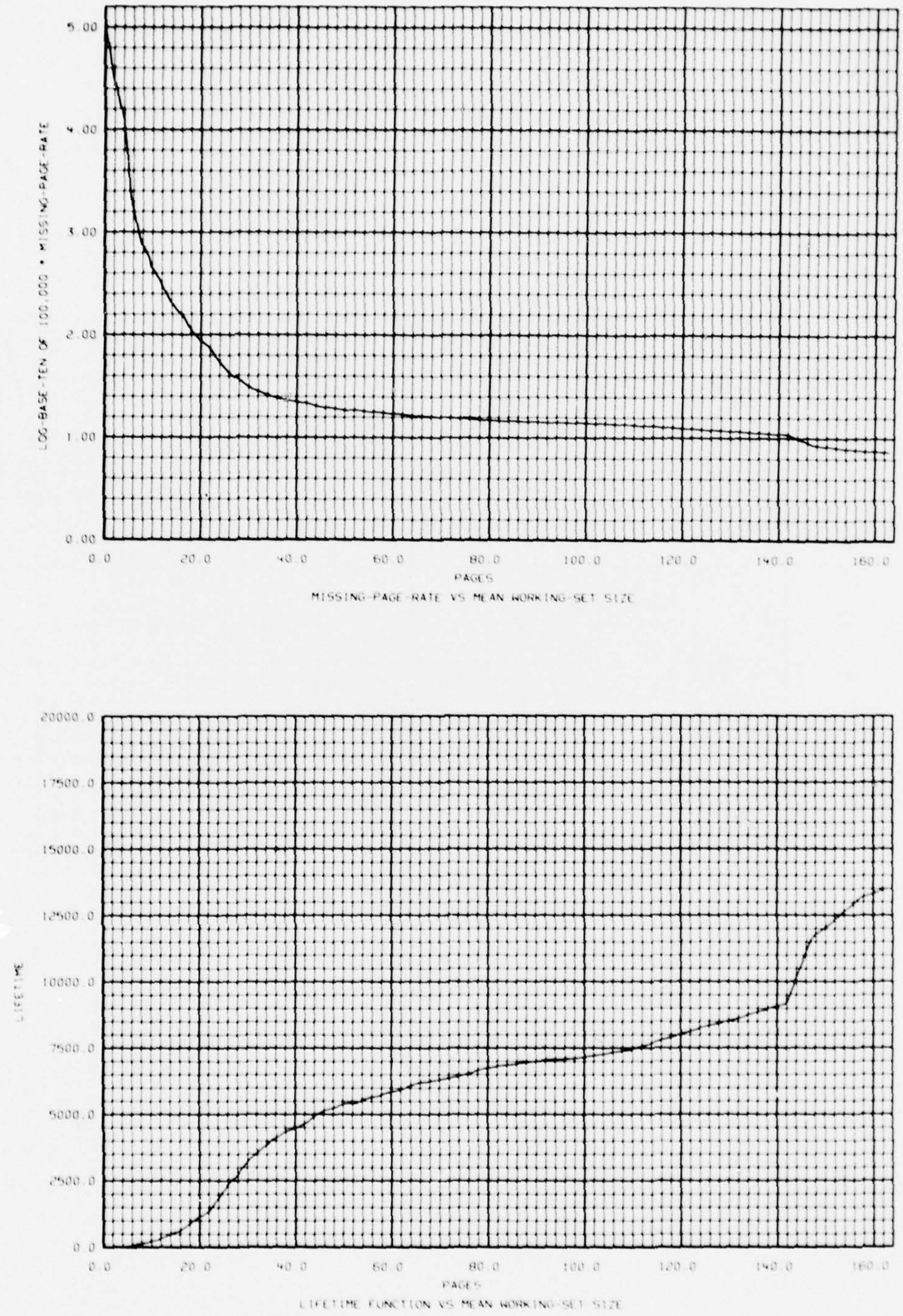
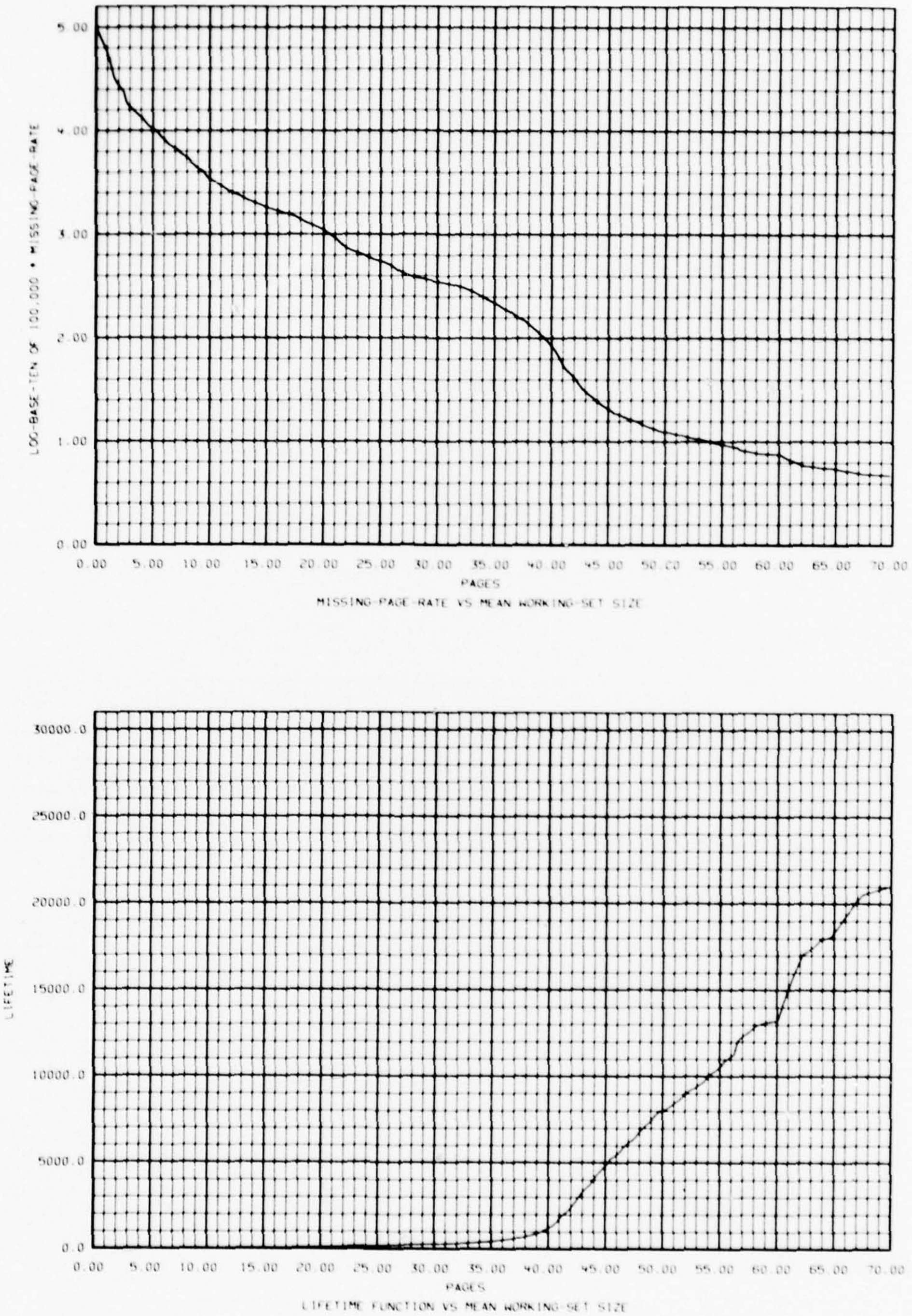


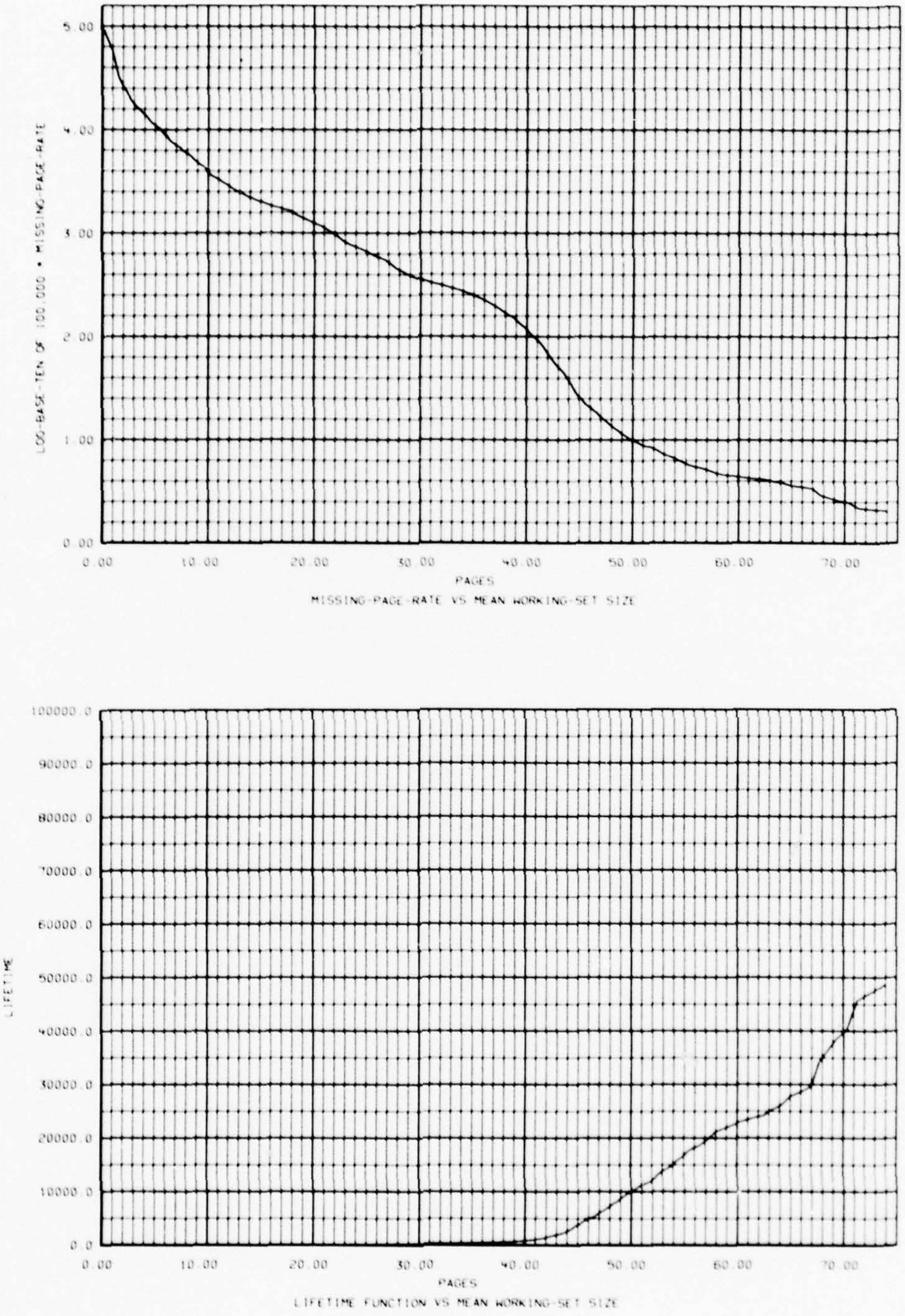
FIGURE B-19 Working-Set Analysis (Continued) for the MUDDLE Compiler



MISSING-PAGE-RATE VS MEAN WORKING-SET SIZE

LIFETIME FUNCTION VS MEAN WORKING-SET SIZE

FIGURE B-20 Working-Set Analysis (Continued) for the MUDDLE Assembler (First Trace)



FIBURE B-21 Working-Set Analysis (Continued) for the MUDDLE Assembler

(Second Trace)

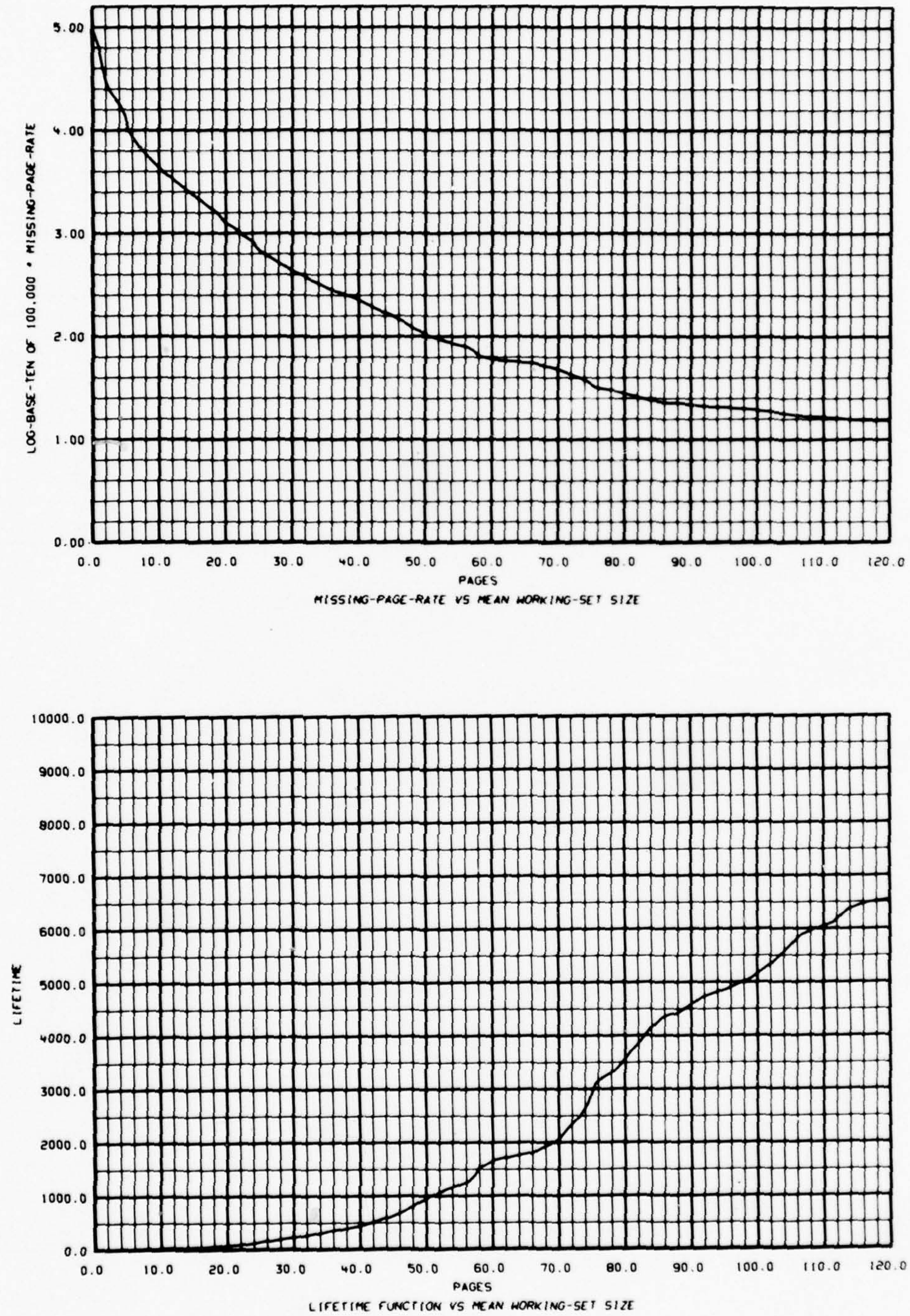
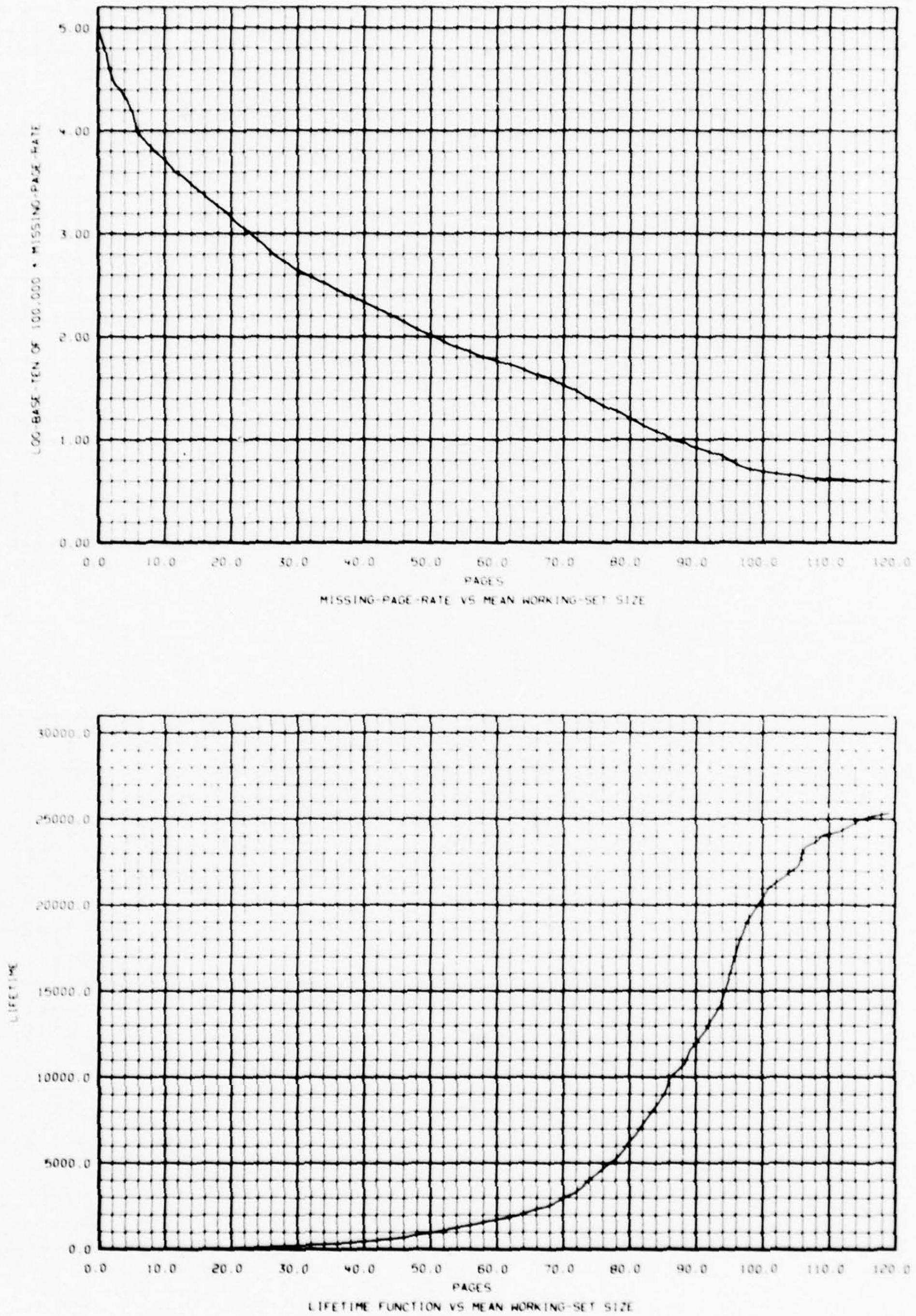


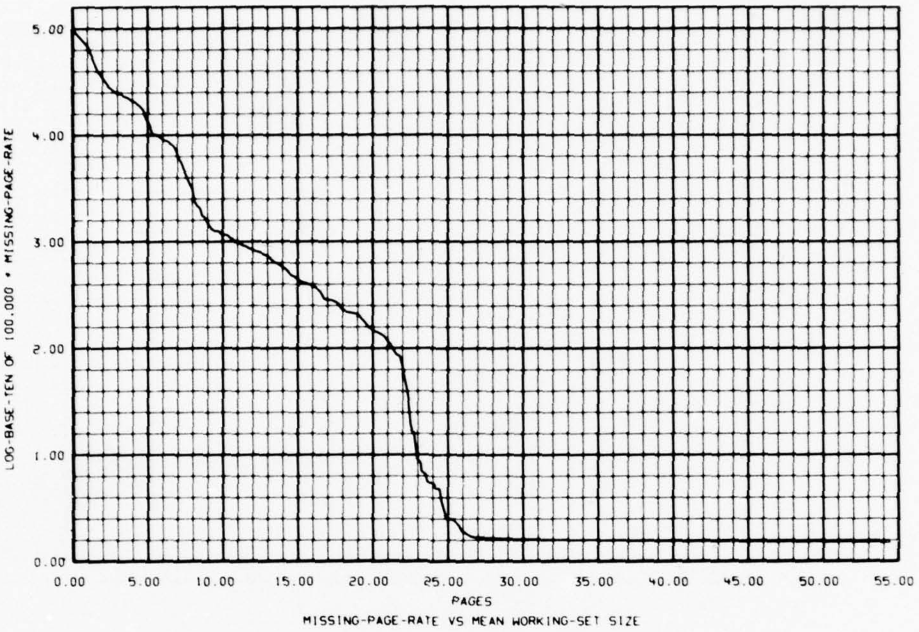
FIGURE B-22 Working- Set Analysis (Continued) for the MIDAS Assembler

(First Trace)



FIBURE B-23 Working-Set Analysis (Continued) for the MIDAS Assembler

(Second Trace)



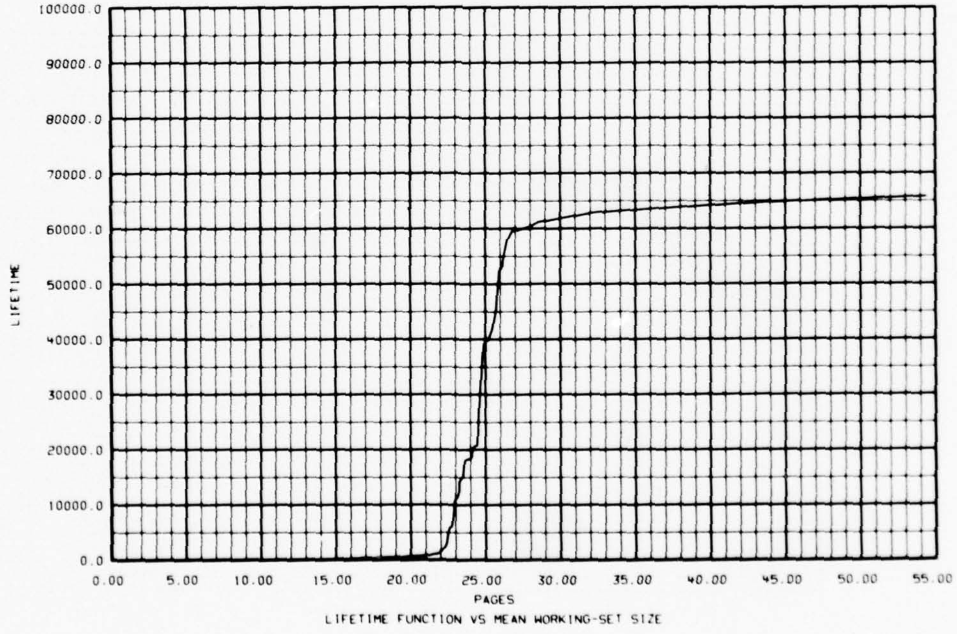


FIGURE B-24 Working-Set Analysis (Continued) for the Text Editor TECO

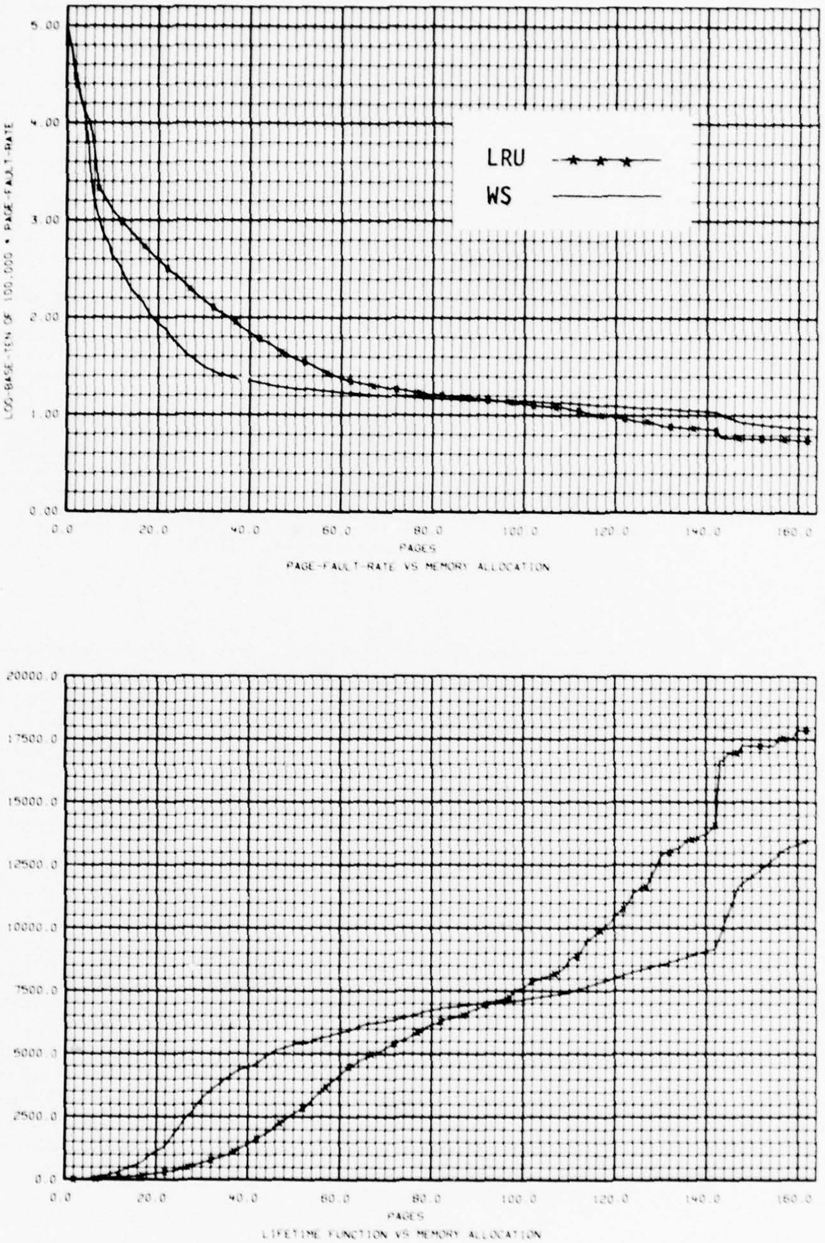


FIGURE B-25 Comparison of LRU and Pure WS for the MUDDLE Compiler

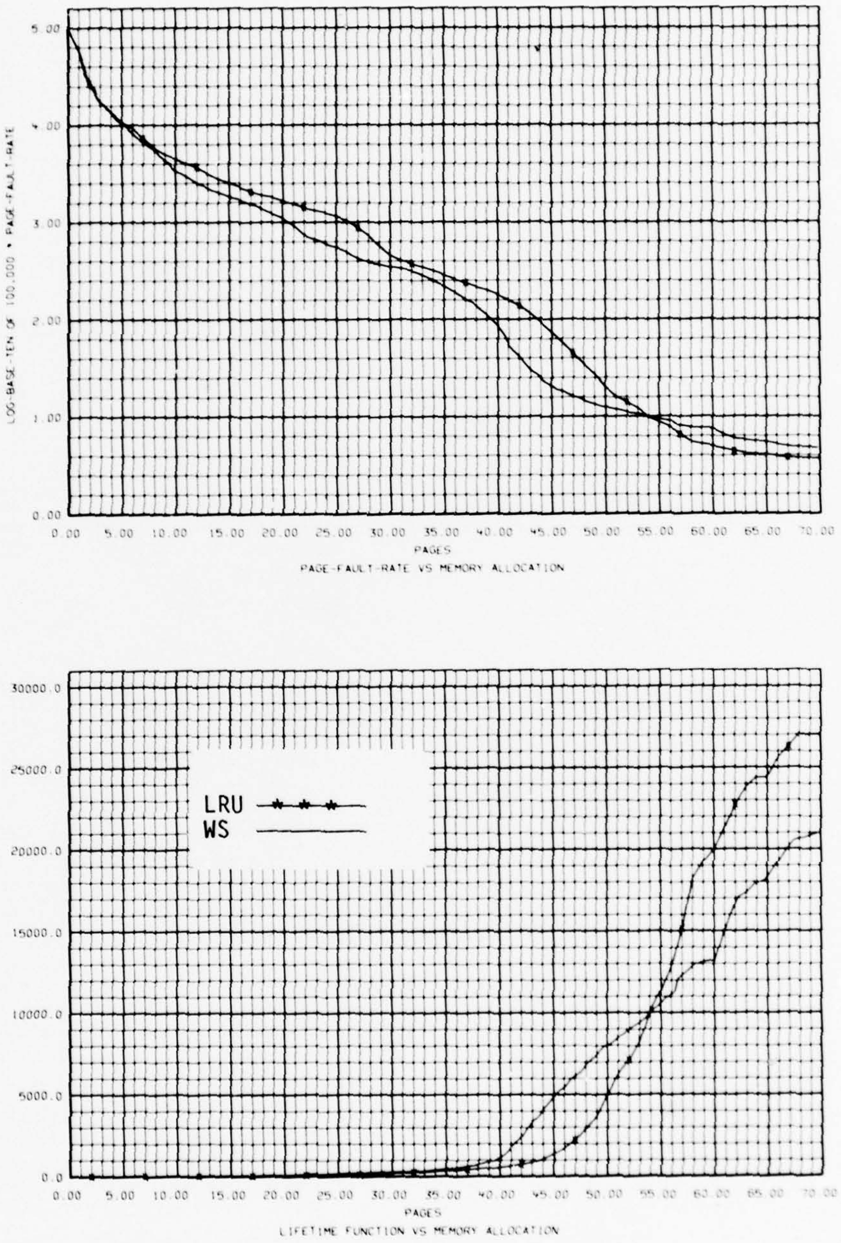


FIGURE B-26 Comparison of LRU and Pure WS for the MUDDLE Assembler

(First Trace)

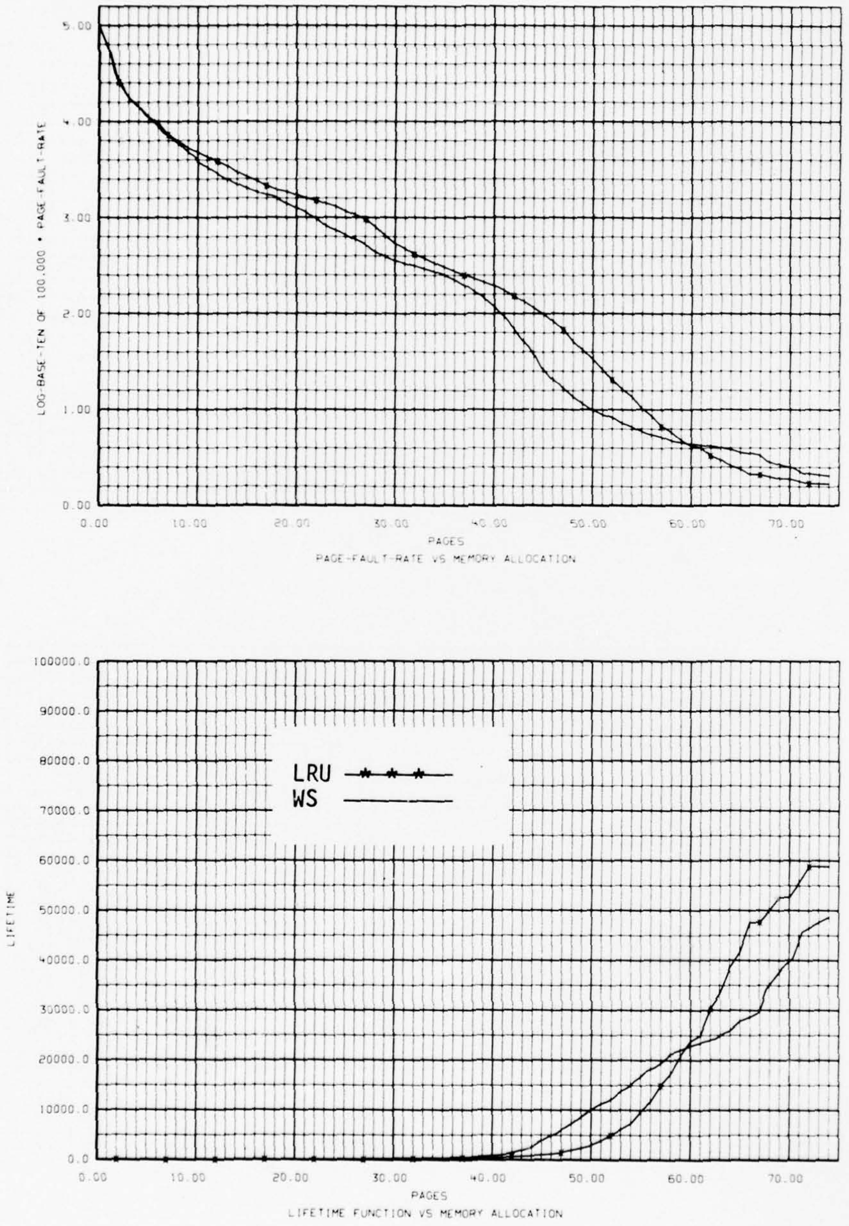


FIGURE B-27 Comparison of LRU and Pure WS for the MUDDLE Assembler

(Second Trace)

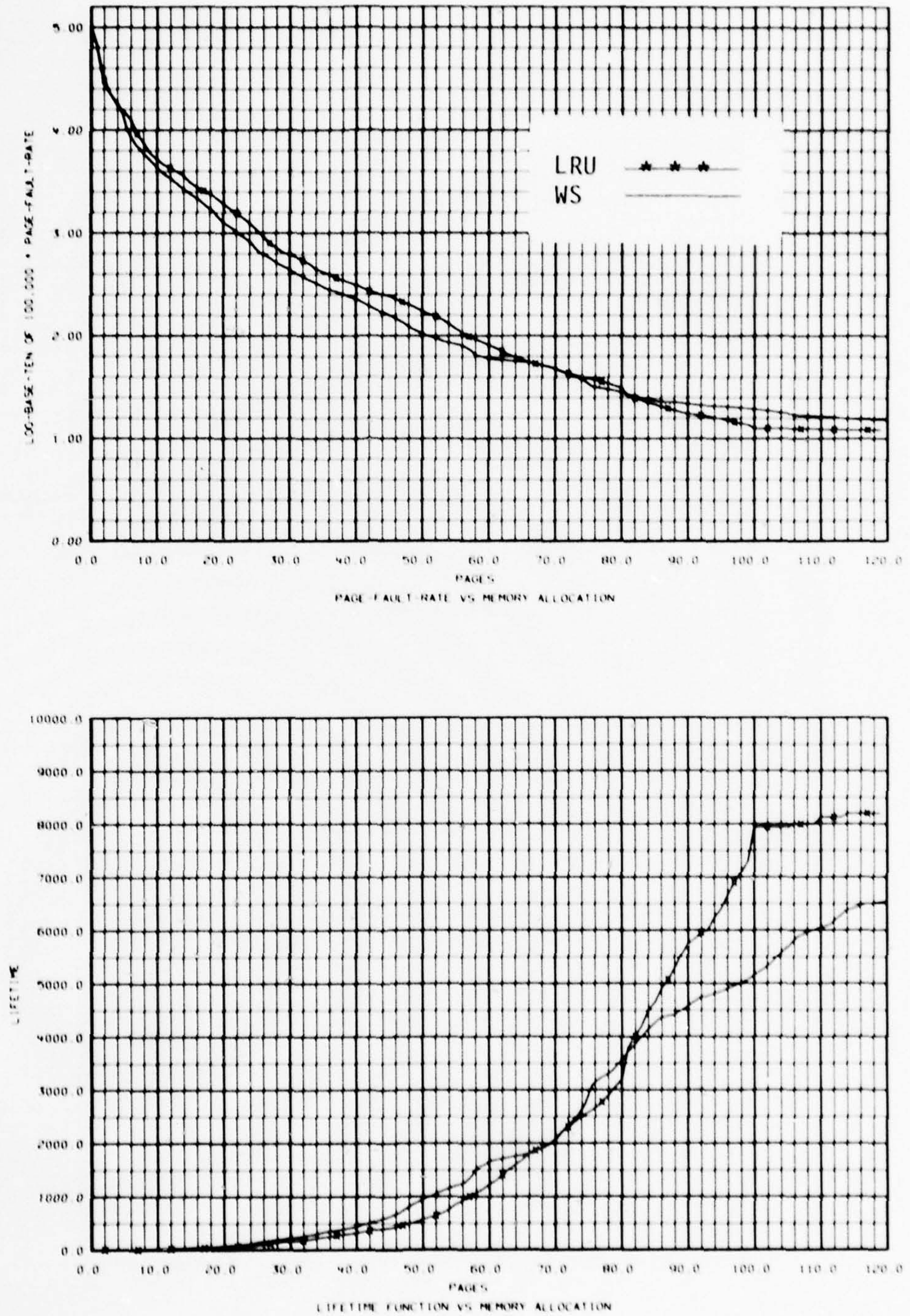


FIGURE B-28 Comparison of LRU and Pure WS for the MIDAS Assembler

(First Trace)

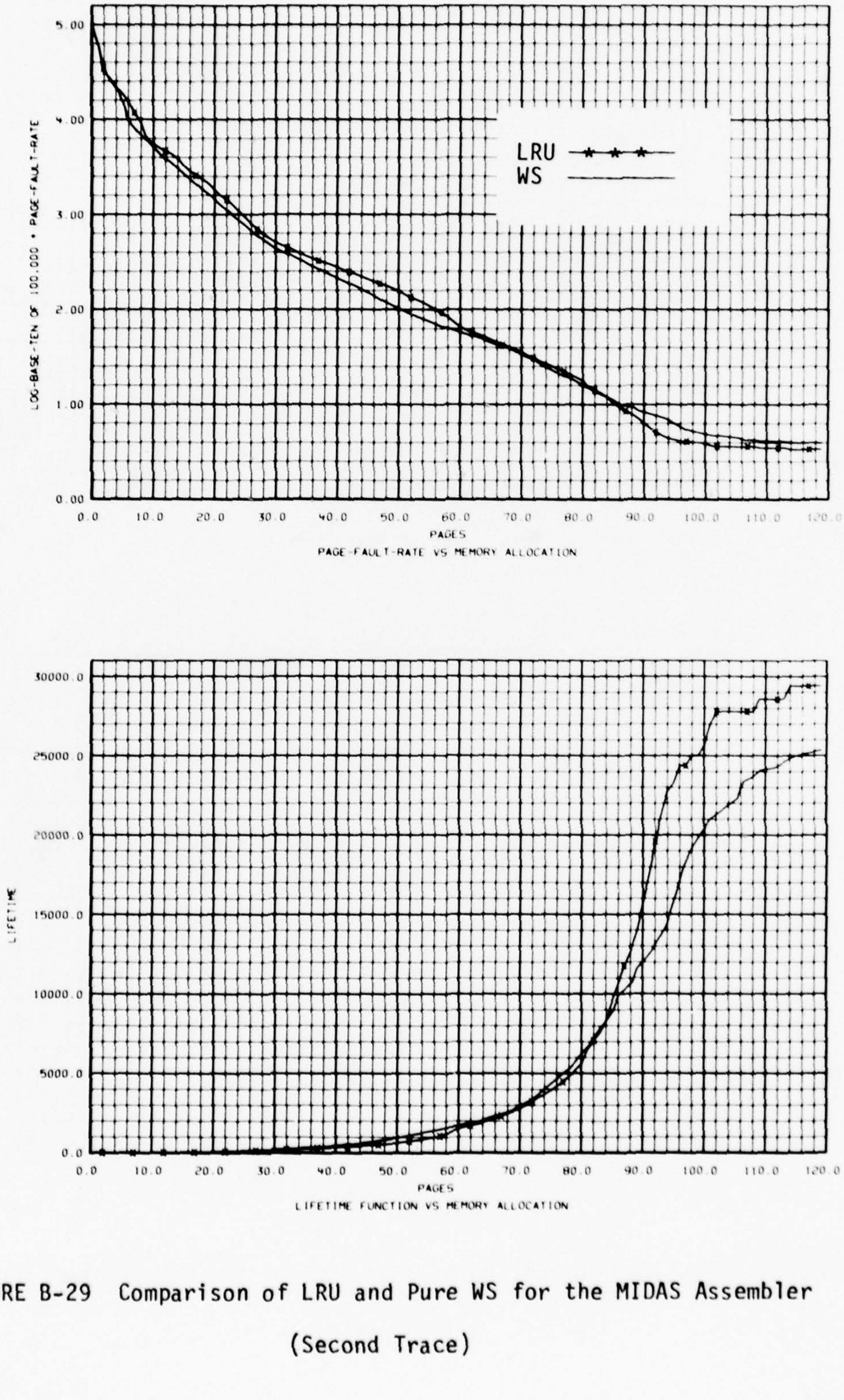
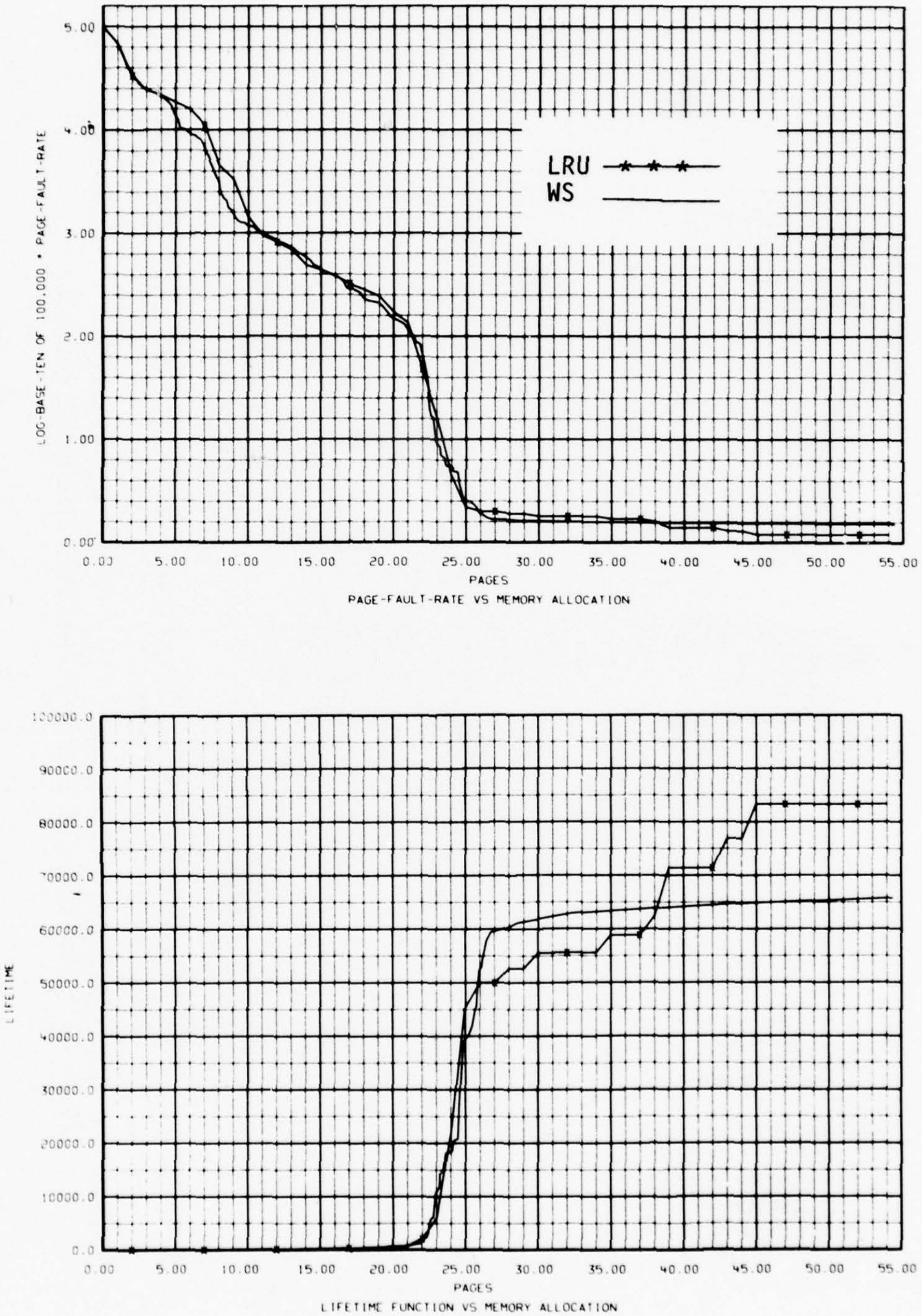


FIGURE B-29 Comparison of LRU and Pure WS for the MIDAS Assembler

(Second Trace)



PAGE-FAULT-RATE VS MEMORY ALLOCATION

LIFETIME FUNCTION VS MEMORY ALLOCATION

FIGURE B-30 Comparison of LRU and Pure WS for the Text Editor TECO

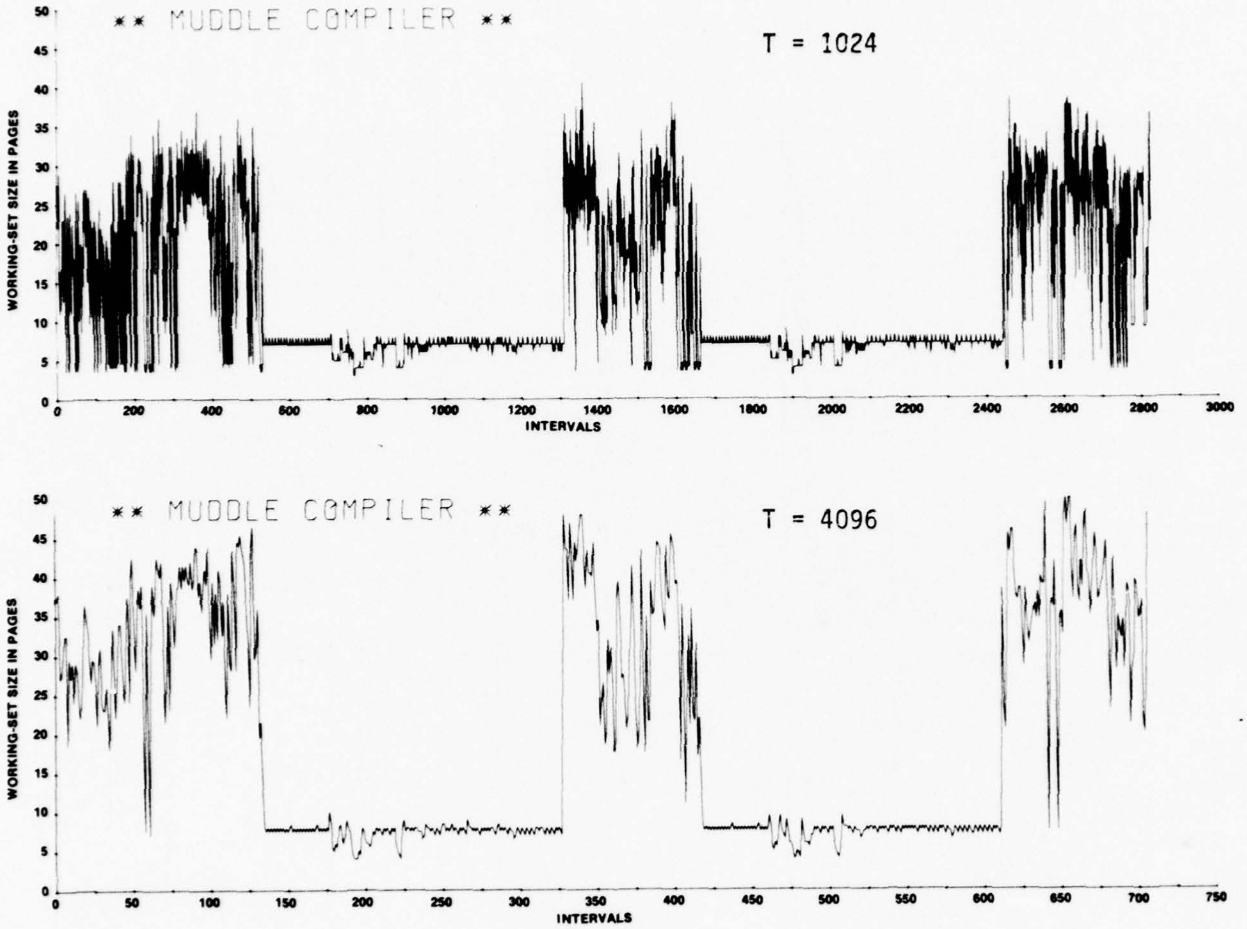
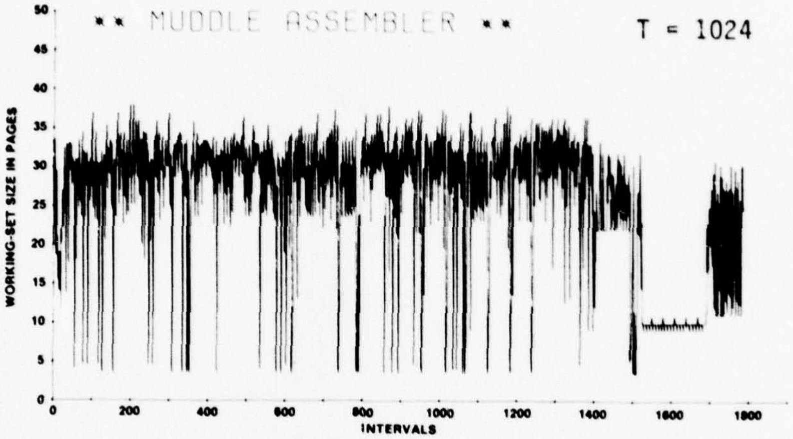


FIGURE B-31 Working-Set Size vs. Time for the MUDDLE Compiler



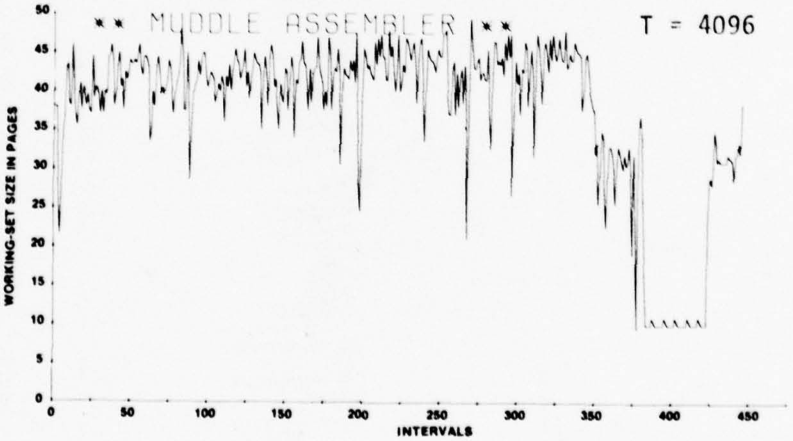


FIGURE B-32 Working-Set Size vs. Time for the MUDDLE Assembler (First Trace)

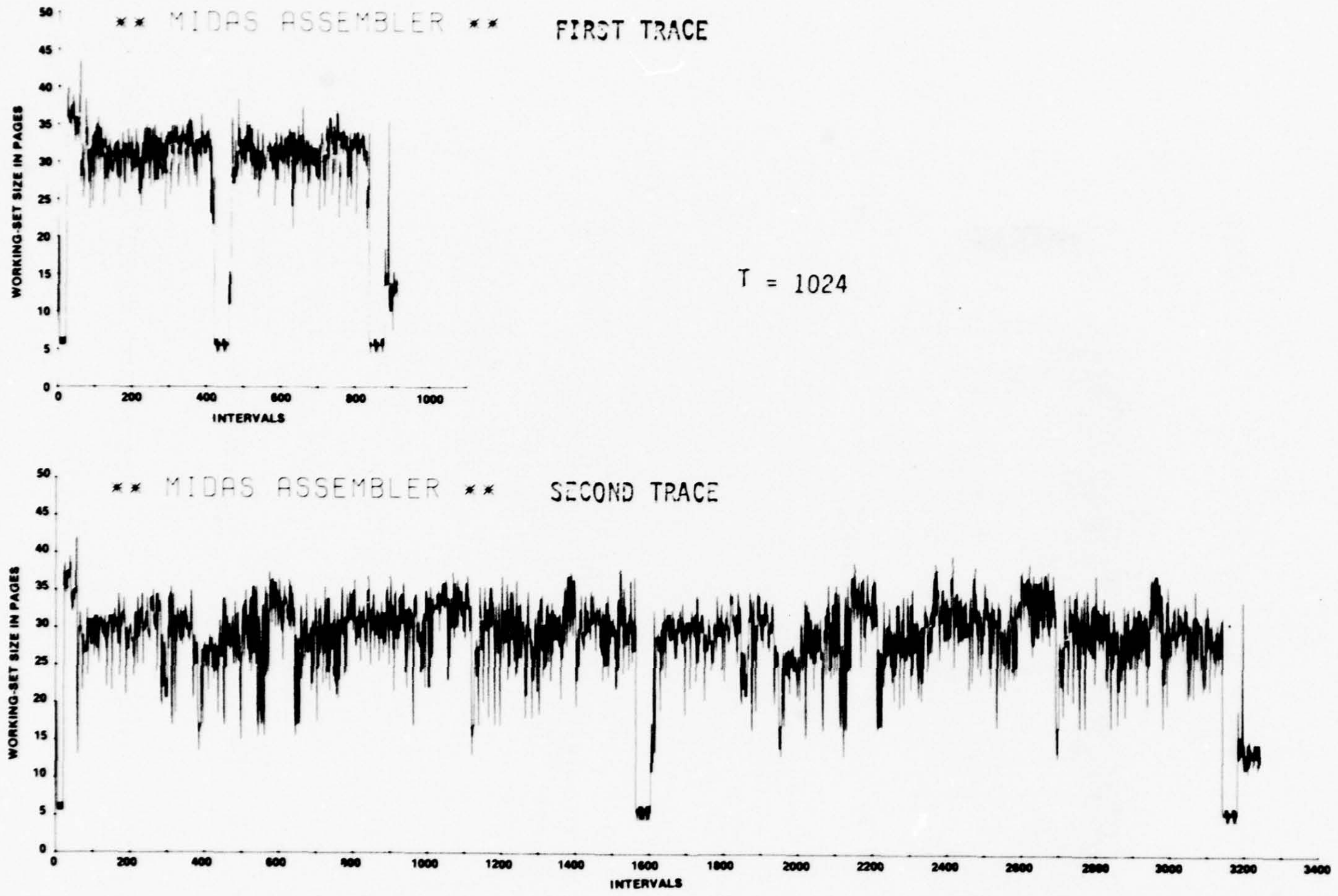


FIGURE B-33 Working-Set Size vs. Time for the MIDAS Assembler

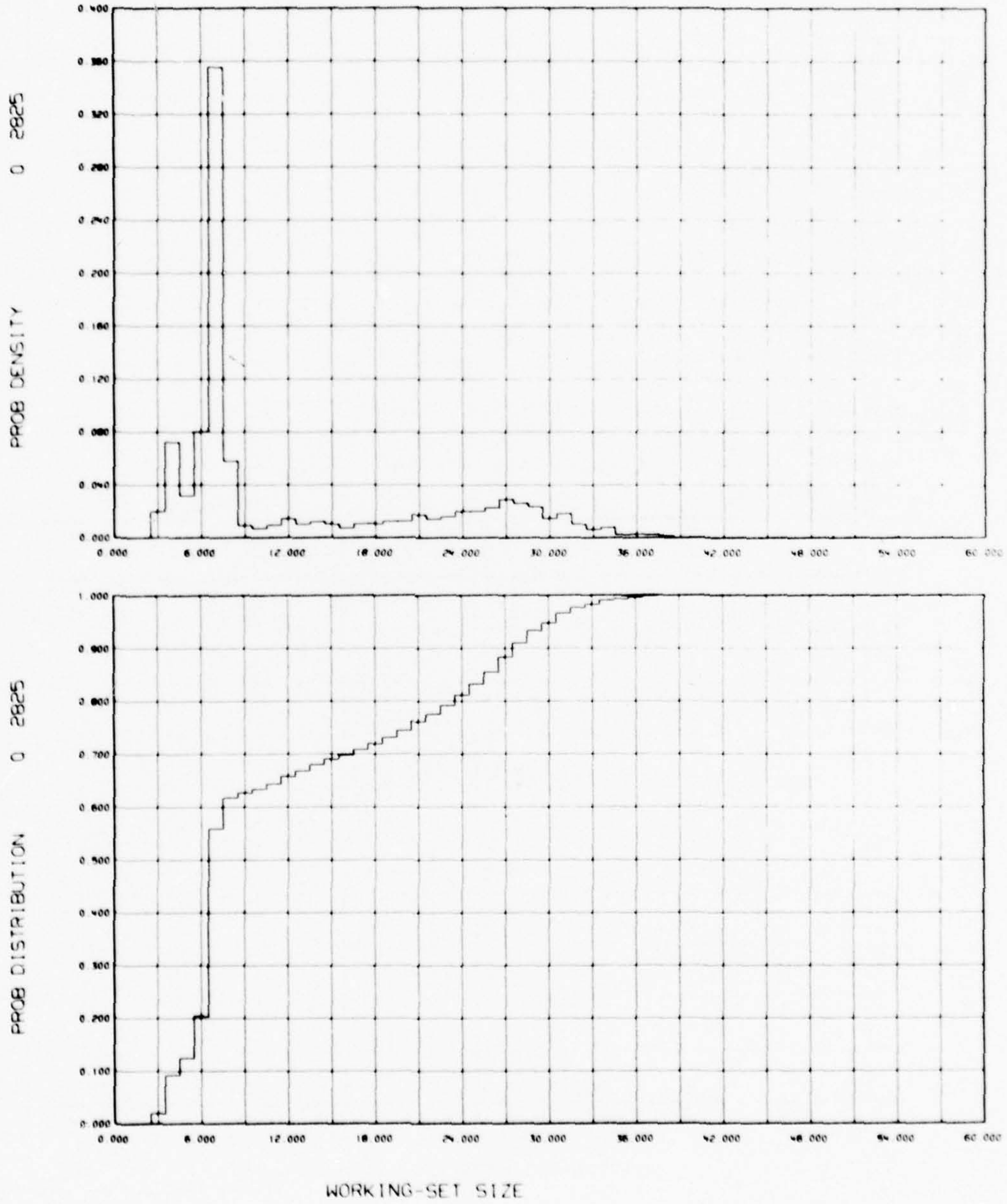


FIGURE B-34 Histogram and Sample CDF of the Working-Set Sizes for the MUDDLE Compiler (T = 1024)

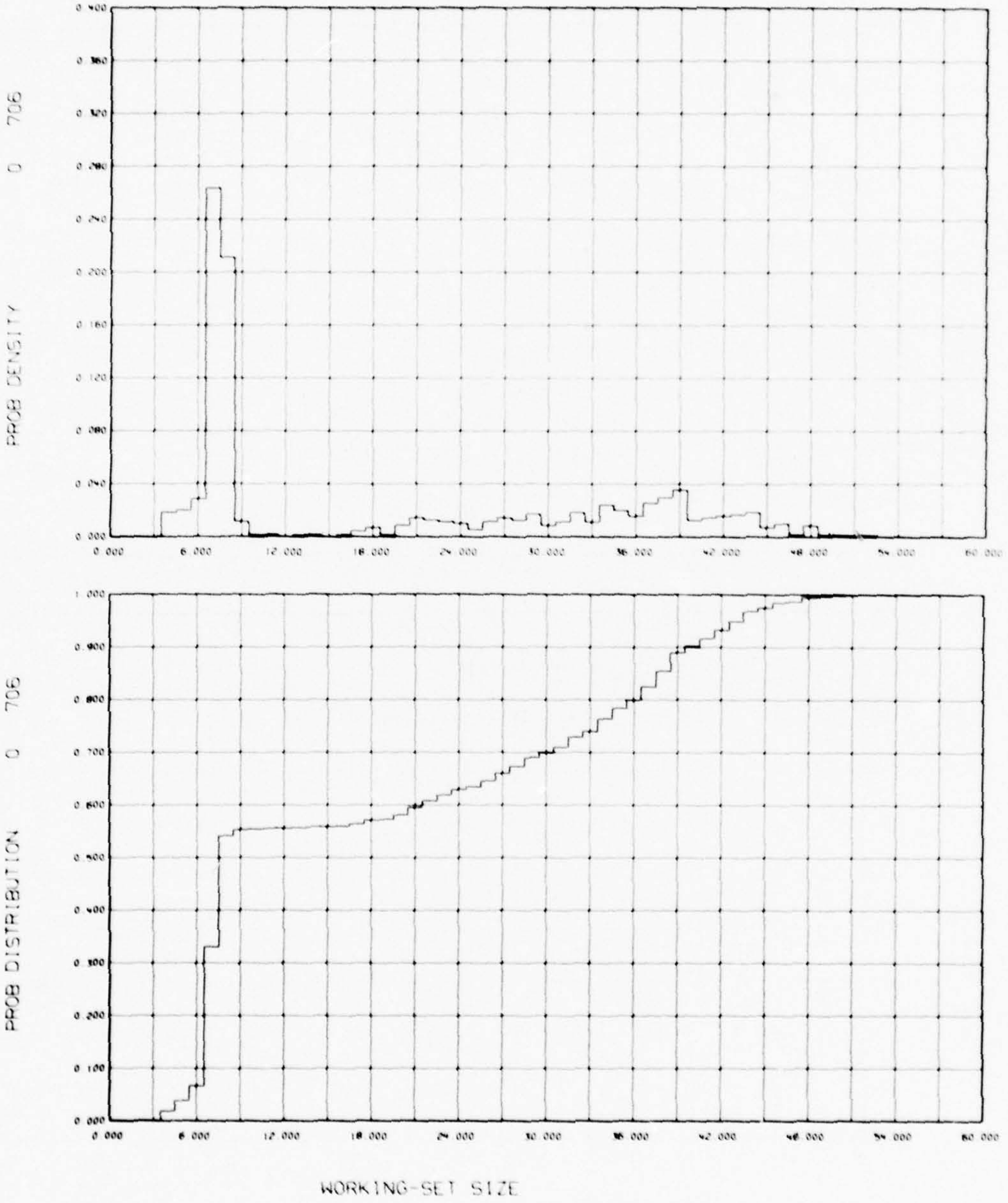


FIGURE B-35 Histogram and Sample CDF of the Working-Set Sizes for the MUDDLE Compiler (T = 4096)

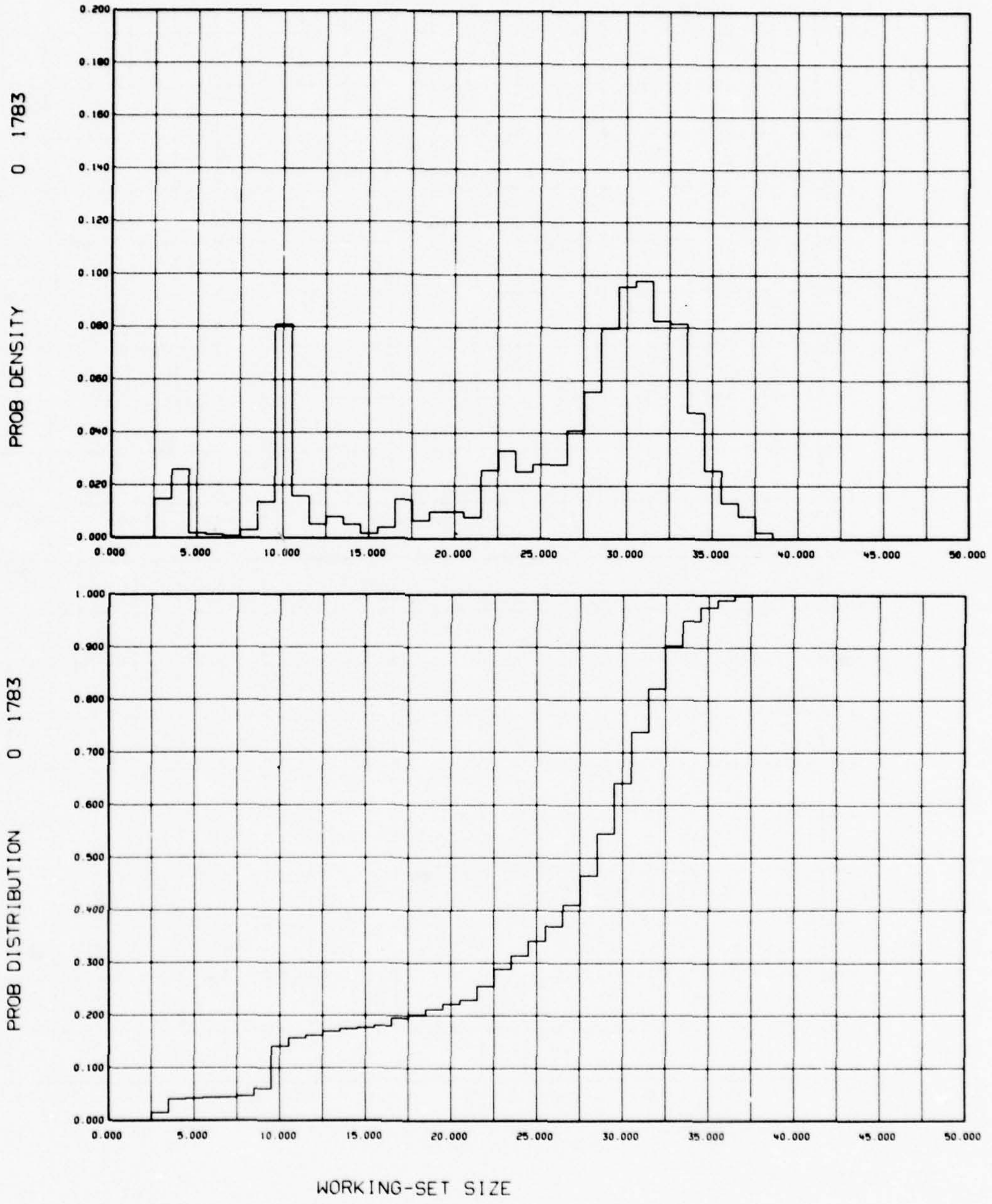


FIGURE B-36 Histogram and Sample CDF of the Working-Set Size for the First Trace of the MUDDLE Assembler (T = 1024)

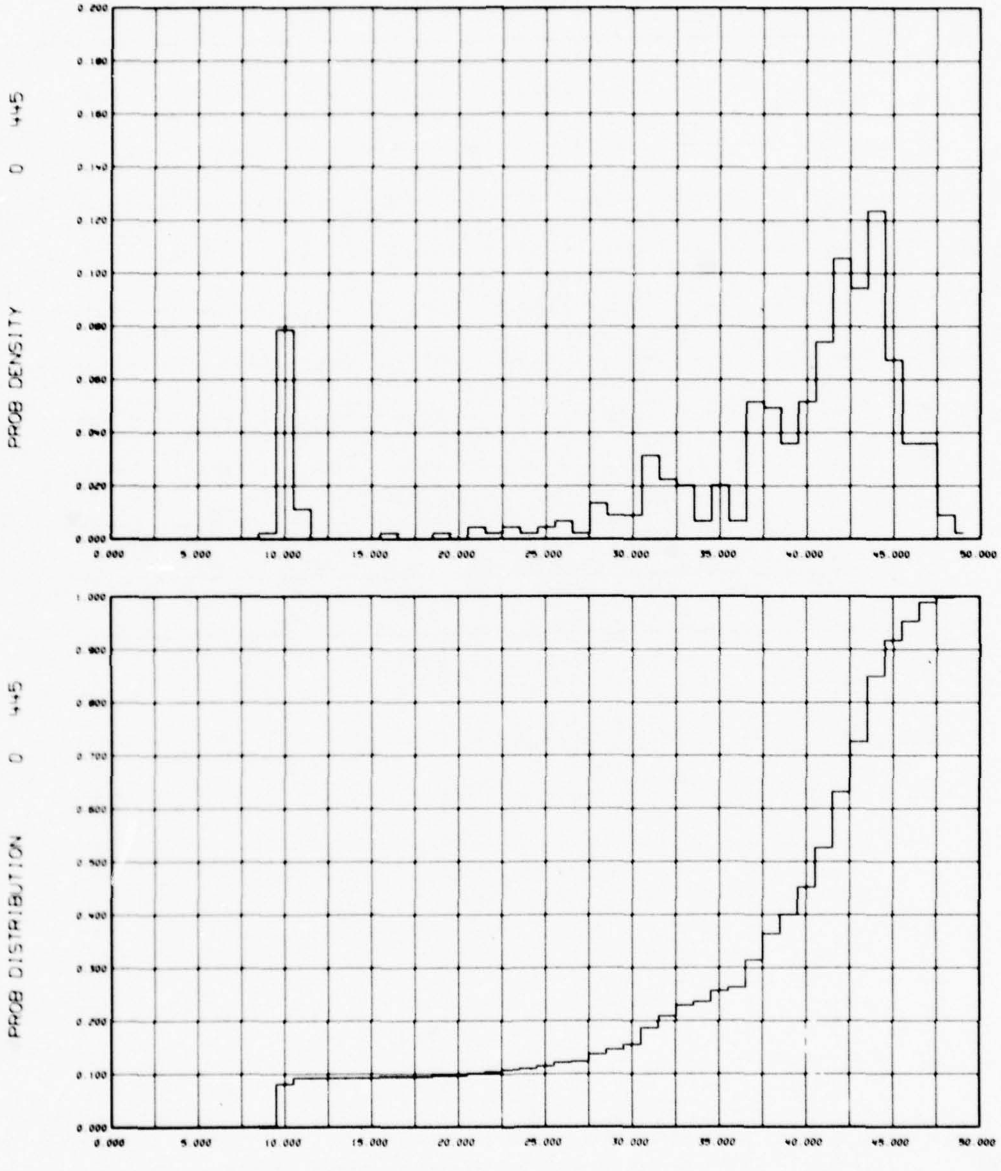


FIGURE B-37 Histogram and Sample CDF of the Working-Set Size for the First Trace of the MUDDLE Assembler (T = 4096)

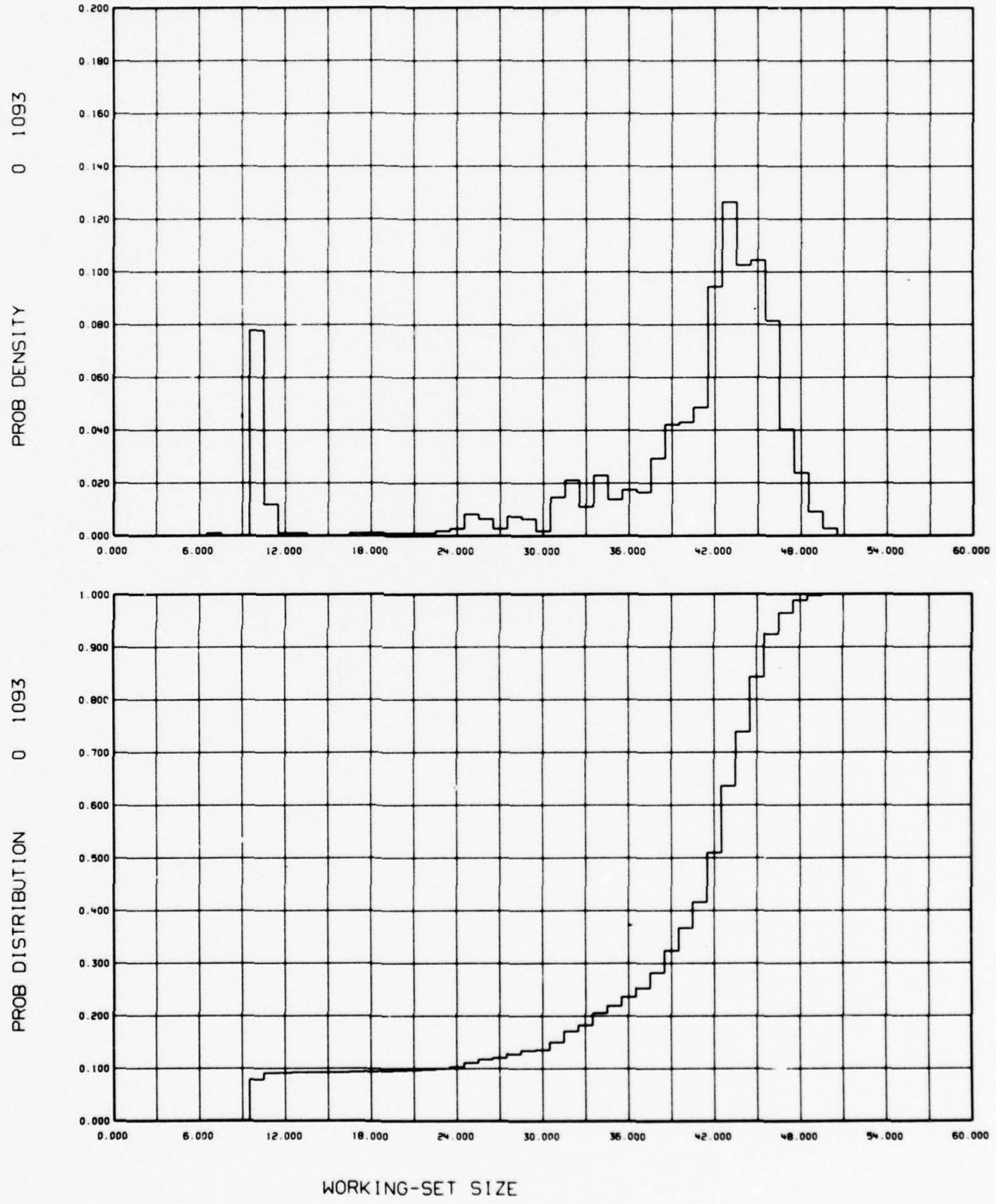


FIGURE B-38 Histogram and Sample CDF of the Working-Set Size for the Second Trace of the MUDDLE Assembler (T = 4096)

REFERENCES

[ADU71] Aho, A. V., P. J. Denning, and J. D. Ullman, "Principles of Optimal Page Replacement", *J. ACM*, Vol. 18 (Jan 1971), pp. 80-93

[ADW68] Abate, J., H. Dubner, and S. B. Weinberg, "Queueing Analysis of the IBM 2314 Disk Storage Facility", *J. ACM*, vol. 15 (Oct 1968), pp. 577-589

[AKK72] Aven, O. I., B. N. Kimel'fel'd, and Ya. A. Kogan, "Control of Multilevel Memory in Computer Systems (Survey)", *Avtomatika i Telemekhanika*, vol. 33, No. 11 (Nov 1972), pp. 138-154

[ArG74] Arnold, C. R., and U. O. Gagliardi, "A State-Space Formulation of the Resource Allocation Problem in Computer Operating Systems", *Proc. Eighth Asilomar Conf. on Circuits, Systems, and Computers*, Pacific Grove, CA (3-5 Dec 1974), pp. 713-722

[ArG76] Arnold, C. R., and S. W. Galley, "Empirical Paged Program Behavior", Submitted to the Int. Symp. on Computer Performance Modelling, Measurement, and Evaluation, Cambridge, MA (29-31 March 1976), Unpublished

[Arn75] Arnold, C. R., "A Control-Theoretic Approach to Memory Management", *Proc. Ninth Asilomar Conf. on Circuits, Systems, and Computers*, Pacific Grove, CA (3-5 Nov 1975), pp. 514-521

[Ash72] Ashany, R., "Application of Control Theory Techniques to Performance Analysis of Computer Systems", *Proc. Sixth Asilomar Conf. on Circuits and Systems*, Pacific Grove, CA (15-17 Nov 1972), pp. 90-101

[Ast70] Astrom, K. J., *Introduction to Stochastic Control Theory*, Academic, New York (1970)

[AtF66] Athans, M., and P. L. Falb, *Optimal Control: An Introduction to the Theory and Its Applications*, McGraw-Hill, New York (1966)

[AvK75] Aven, O. I., and Ya. A. Kogan, "Stochastic Control of Paging in a Two-Level Computer Memory", *Automatica*, vol. 11 (1975), pp. 309-311

[BeD62] Bellman, R. E., and S. E. Dreyfus, Applied Dynamic Programming, Princeton Univ. Press, Princeton, NJ (1962)

[Bel66] Belady, L. A., "A Study of Replacement Algorithms for Virtual-Storage Computers", IBM Systems J., vol. 5 (1966), pp. 78-101

[BGLLP74] Badel, M., E. Gelenbe, J. Lenfant, J. Leroudier, and D. Potier, "Adaptive Optimization of the Performance of a Virtual Memory Computer", in Computer Architectures and Networks, E. Gelenbe and R. Mahl (Eds.), North Holland Pub. Co. (1974), pp. 1-26

[BGLP75] Badel, M., E. Gelenbe, J. Leroudier, and D. Potier, "Adaptive Optimization of a Time-Sharing System's Performance", Proc. IEEE, vol. 63 (June 1975), pp. 958-965

[B-IG74] Ben-Israel, A., and T. N. E. Greville, Generalized Inverses: Theory and Applications, Wiley-Interscience, New York (1974)

[BlR72] Blevins, P. R., and C. V. Ramamoorthy, "Aspects of a Dynamically Adaptive Operating System", Tech. Report No. 132, Univ. Texas, Austin, TX (31 Jul 1972)

[BoK74] Boguslavskii, L. B., and Ya. A. Kogan, "An Analysis of Page Replacement Algorithms in a Two-Level Digital Computer", Avtomatika i Telemekhanika, vol. 35, No. 11 (Nov 1974), pp. 129-136

[B-R60] Bharucha-Reid, A. T., Elements of the Theory of Markov Processes and Their Applications, McGraw-Hill, New York (1960)

[BrH69] Bryson, A. E., Jr., and Yu-Chi Ho, Applied Optimal Control, Ginn, Waltham, MA (1969)

[Bry75] Bryant, P., "Predicting Working Set Size", IBM J. Res. Develop., vol. 19 (May 1975), pp. 221-229

[BuJ68] Bucy, R. S., and P. D. Joseph, Filtering for Stochastic Processes with Applications to Guidance, Interscience, New York (1968)

[BuS74] Buzen, J. P., and A. W. C. Shum, "Structural Considerations for Computer System Models", Proc. Eighth Annual Princeton Conference on Information Sciences and Systems (March 1974)

[Buz71a] Buzen, J. P., "Analysis of System Bottlenecks Using a Queueing Network Model", Proc. ACM-SIGOPS Workshop on System Performance Evaluation, Cambridge, MA (April 1971), pp. 82-103

[Buz71b] Buzen, J. P., "Queueing Network Models of Multiprogramming", Ph.D. Dissertation, Div. Engrg. and Applied Physics, Harvard University, Cambridge, MA (May 1971)

[Buz73] Buzen, J., "Optimally Balancing of I/O Requests to Sector-Scheduled Drums", Proc. Seventh Annual ACM-ASA Symp. on Computer Science and Statistics, Iowa State Univ., Ames, IA (Oct 1973), pp. 130-138

[Che65] Chestnut, H., System Engineering Tools, Wiley, New York (1965)

[Che73] Chen, P. P. S., "Optimal File Allocation in Multi-Level Storage Systems", AFIPS Conf. Proc., vol. 42 (June 1973 NCC), pp. 277-282

[ChO72] Chu, W. W., and H. Opderbeck, "The Page Fault Frequency Replacement Algorithm", AFISP Conf. Proc., vol. 41 (1972 FJCC), pp. 597-609

[ChO74] Chu, W. W., and H. Opderbeck, "Performance of Replacement Algorithms with Different Page Sizes", Computer, vol. 7, No. 11, (Nov 1914), pp. 14-21

[CoD73] Coffman, E. G., Jr., and P. J. Denning, Operating System Theory, Prentice-Hall, Englewood Cliffs, NJ, (1973)

[Cof76] Coffman, E. G., Jr., (Ed), Computer and Job-Shop Scheduling Theory, Wiley-Interscience, New York (1976)

[CoM65] Cox, D. R., and H. D. Miller, The Theory of Stochastic Processes, Wiley, New York (1965)

[CoR72] Coffman, E. G., Jr., and T. A. Ryan, Jr., "A Study of Storage Partitioning Using a Mathematical Model of Locality", Comm. ACM, vol. 15 (March 1972), pp. 185-190

[CoV68] Coffman, E. G., and L. C. Varian, "Further Experimental Data on the Behavior of Programs in a Paging Environment", Comm. ACM, vol. 11 (July 1968), pp. 471-474

[Cou75] Courtois, P. J., "Decomposability, Instabilities, and Saturation in Multiprogramming Systems", Comm. ACM, vol. 18, (July 1975), pp. 371-377

[Cou77] Courtois, P. J., DECOMPOSABILITY Queueing and Computer System Applications, Academic, New York (1977)

[Cox62] Cox, D. R., Renewal Theory, Methuen, London (1962)

[Dan75] Dantzig, G. B., "Discrete-Variable Extremum Problems", Operations Research, vol. 5 (April 1957), pp. 266-277

[DeE71] Denning, P. J., and B. A. Eisenstein, "Statistical Methods in Performance Evaluation", ACM SIGOPS Workshop on System Performance Evaluation, Harvard Univ., Cambridge, MA (5-7 April 1971), pp. 284-307

[DeG75] Denning, P. J., and G. S. Graham, "Multiprogrammed Memory Management", Proc. IEEE, vol. 63 (June 1975), pp. 924-939

[DeK75] Denning, P. J., and K. C. Kahn, "A Study of Program Localities and Lifetime Functions", Proc. Fifth Symp.on Operating Systems Principles (Nov 1975), pp. 207-216

[Den67] Denning, P. J., "Effects of Scheduling on File Memory Operations", AFIPS Conf. Proc., vol. 30 (1967 SJCC), pp. 9-21

[Den68a] Denning, P. J., "Resource Allocation in Multiprocess Computer Systems", MIT Ph.D. thesis - pub. as MIT Project MAC Report MAC-TR-50 (May 1968)

[Den68b] Denning, P. J., "The Working Set Model for Program Behavior", Comm. ACM, vol. 11 (May 1968), pp. 323-333

[Den68c] Denning, P. J., "Trashing: Its Causes and Prevention", AFIPS Conf. Proc., vol. 33 (1968 FJCC), pp. 915-922

[Den69] Denning, P. J., "Equipment Configuration in Balanced Computer Systems", IEEE Trans. Computers, vol. C-18 (Nov 1969) pp. 1008-1012

[Den70] Denning, P. J., "Virtual Memory", Comp. Surveys, vol. 2 (Sept 1970), pp. 153-189

[Den72] Denning, P. J., "On Modelling Program Behavior", AFIPS Conf. Proc., vol. 40, (1972 SJCC), pp. 937-944

[DeS72] Denning, P. J., and S. C. Schwartz, "Properties of the Working-Set Model", Comm. ACM, vol. 15 (March 1972), pp. 191-198

[DSS72] Denning, P. J., J. R. Spirn and J. E. Savage, "Some Thoughts about Locality in Program Behavior", Proc. Symp. on Computer-Communications Networks and Teletraffic, Polytechnic Inst. of Brooklyn, New York (4-6 April 1972), pp. 101-112

[EGHKN69] Eastlake, D. E., R. Greenblatt, J. Holloway, T. Knight, and S. Nelson, "ITS 1.5 Reference Manual", Memo No. 161A, Artificial Intelligence Lab, MIT, Cambridge, MA (July 1969)

[Eve63] Everett, H., "Generalized Lagrange Multiplier Methods for Solving Problems of Optimum Allocation of Resources", Operations Research, vol. 11 (1963), pp. 399-417

[Eyk74] Eykoff, P., System Identification: Parameter and State Identification, Wiley-Interscience, New York (1974)

[Fra69] Frank, H., "Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems", J. ACM, vol. 16 (Oct 1969), pp. 602-620

[Ful72] Fuller, S. H., "An Optimal Drum Scheduling Algorithm", IEEE Trans. Computers, vol. C-21 (Nov 1972), pp. 1153-1165

[Ful75] Fuller, S. H., Analysis of Drum and Disk Storage Units, Springer-Verlag, Berlin (1975)

[GaG74] Galley, S. W. and R. P. Goldberg, "Software Debugging: The Virtual Machine Approach", Proc. ACM National Conf., San Diego, CA (1974), pp. 395-401

[Gal71] Galley, S. W. "Debugging Using ESP - Execution Simulator and Presenter", Document No. SYS.09.01, Programming Technology Division, Project MAC, MIT, Cambridge, MA (Nov 1971)

[GaP75] Galley, S. W., and G. Pfister, "The MDL Language", Document No. SYS.11.01, Programming Technology Division, Project MAC, MIT, Cambridge, MA (9 Sep 1975)

[GaS71] Gaver, D. P., and G. S. Shedler, "Control Variable Methods in the Simulation of a Model of a Multiprogrammed Computer System", Naval Res. Logist. Quart., Vol, 18 (Dec 1971), pp. 435-450

[Gel74] Gelb, A., (Editor), Applied Optimal Estimation, MIT Press, Cambridge, MA (1974)

[GPBL73] Gelenbe, E., D. Potier, A. Brandwajn, and J. Lenfant, "Gestion Optimale D'un Ordinateur Multiprogramme a Memorie Virtuelle", Proc. Fifth Conf. Optimization Techniques, Springer-Verlag, Berlin (1973)

[Gol74] Golomb, S. W., Shift Register Sequences, Holden-Day, San Franscisco, CA (1974)

[Hal60] Halmos, P. R., Naive Set Theory, Van Nostrand, Princeton, NJ (1960)

[Ham73] Hamlet, R. G., "Efficient Multipgramming Resource Allocation and Accounting", Comm. ACM, vol. 16 (June 1973), pp. 337-343

[Hat72] Hatfield, D. J., "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance", IBM J. Res. Develop., vol. 16 (Jan 1972), pp. 58-66

[Hil56] Hildebrand, F. B., Introduction to Numerical Analysis, McGraw-Hill, New York (1956)

[How60] Howard, R. A., Dynamic Programming and Markov Processes, MIT Press, Cambridge, MA (1960)

[How64] Howard, R. A., "System Analysis of Semi-Markov Processes", IEEE Trans. Military Electronics, vol. MIL-8 (April 1964), pp. 114-124

[HR-R74] Henderson, G., and J. Rodriguez-Rosell, "The Optimal Choice of Window Sizes for Working Set Dispatching", ACM Performance Evaluation Review, vol. 3 (Dec 1974), pp. 10-33

[Hu69] Hu, T. C., Integer Programming and Network Flows, Addison-Wesley, Reading, MA (1969)

[InK74] Ingargiola, G., and J. F. Korsh, "Finding Optimal Demand Paging Algorithms", J. ACM, vol. 21 (Jan 1974), pp. 40-53

[Jai78] Jain, R. K., "Control-Theoretic Formulation of Operating Systems Resource Management Policies", Ph.D. Thesis, Harvard University, Cambridge, MA (May 1978)

[KaB61] Kalman, R. E., and R. S. Bucy, "New Results in Linear Filtering and Prediction Theory", Trans. ASME (J. Basic Engineering), vol. 83, part D (March 1961), pp. 95-108

[Kal60] Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems", Trans. ASME (J. Basic Engineering), vol. 82, part D (March 1960), pp. 35-45

[Kam75] Kameda, H., "The Analysis of an Adaptive Workload Balancing Strategy in Computing System Resources Management", Int. J. Comp. and Information Sciences, vol. 4 (1975), pp. 295-306

[Kle76] Kleinrock, L., Queueing Systems, Vol. II: Computer Applications, Wiley-Interscience, New York (1976)

[Kog73] Kogan, Ya. A., "Markov Models of Page Replacement in a Two-Level (Virtual) Computer Memory", Avtomatika i Telemekhanika, vol. 34, no. 4 (April 1973), pp. 146-154

[Kus71] Kushner, H. J., Introduction to Stochastic Control, Holt, Rinehart and Winston, New York (1971)

[LAG68] Lee, T. H., G. E. Adams, and W. M. Gains, Computer Process Control: Modeling and Optimization, Wiley, New York (1968)

[Lan56] Lanczos, C., Applied Analysis, Prentice-Hall, Englewood Cliffs, NJ (1956)

[LaW66] Lawler, E. L., and D. E. Wood, "Branch-and-Bound Methods: A Survey", Operations Research, vol. 14 (1966), pp. 699-719

[Lee60] Lee, Y. W., Statistical Theory of Communications, Wiley, New York (1960)

[Lee64] Lee, R. C. K., Optimal Estimation, Identification, and Control, MIT Press, Cambridge, MA (1964)

[LeS73] Lewis, P. A. W., and G. S. Shedler, "Empirically Derived Micromodels for Sequences of Page Exceptions", IBM J. Res. Develop., vol. 17 (March 1973), pp. 86-100

[Lew72] Lewis. P. A. W., (Editor), Stochastic Point Processes, Wiley, New York (1972)

[Lew75a] Lew, A., "Optimal Resource Allocation and Scheduling Among Parallel Processes", in Parallel Processing, Tse-Yun Feng, Ed., Springer-Verlag, Berlin, (1975), pp. 187-202

[Lew75b] Lew, A., "Optimal Control of Demand-Paging Systems", Proc. Third Milwaukee Symp. on Automatic Computation and Control, Milwaukee, WI, (18-19 April 1975), pp. 355-360

[Lew75c] Lew, A., "On Optimal Demand-Paging Algorithms", Proc. Second USA-JAPAN Computer Conf., (26-28 Aug 1975), pp. 596-600

[LeY71] Lewis, P. A. W., and P. C. Yue, "Statistical Analysis of Program Reference Patterns in a Paging Environment", Proc. 1971 IEEE Computer Society Conf., Boston, MA (Sept 1971), pp. 133-134

[Lip68] Liptay, J. S., "The Cache", IBM Systems J., vol. 7 (1968), pp. 15-21

[MaB75] Madison, A. W., and A. P. Batson, "Characteristics of Program Localities", Proc. Fifth ACM-SIGOPS Conf. (Nov 1975) pp. 65-73; also in Comm. ACM, vol. 19 (May 1976), pp. 285-294

[Mad73] Madnick, S. E., "Storage Hierarchy Systems", MIT Project MAC Report TR-105, Cambridge, MA (April 1973)

[Mad74] Madnick, S. E., and J. J. Donovan, Operating Systems, McGraw-Hill, New York (1974)

[Mah70] Mahl, R., "An Analytical Approach to Computer System Scheduling", Tech. Report No. UTECH-CSC-70-100, Univ. Utah, Salt Lake City (June 1970)

[MaR76] Mahmoud, S., and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks", ACM Trans. Database Systems, vol. 1 (March 1976), pp. 66-78

[McG74] McGarty, T. P., Stochastic Systems and State Estimation, Wiley-Interscience, New York (1974)

[MGST70] Mattson, R. L., J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies", IBM Systems J., vol. 9, (1970), pp. 78-117

[Moo74] Moon, D. A., MACLISP Reference Manual, MIT Project MAC, Cambridge, MA, (April 1974)

[New71] Newell, G. F., Application of Queueing Theory, Chapman and Hall, London (1971)

[Nie68] Nielsen, N. R., "Flexible Pricing: An Approach to the Allocation of Computer Resources", AFIPS Conf. Proc., vol. 33 (1968 FJCC), pp. 521-531

[Nie70] Nielsen, N. R., "The Allocation of Computer Resources - Is Pricing the Answer?", Comm. ACM, vol. 13 (Aug 1970), pp. 467-474

[OCO72] Oliver, N., W. W. Chu, and H. Opderbeck, "Measurement Data on the Working Set Replacement Algorithm and Their Applications", Proc. Symp. on Computer-Communications Networks and Teletraffic, Polytechnic Inst. of Brooklyn, New York (4-6 April 1972), pp. 113-124

[OdS72] Oden, P. H., and G. S. Shedler, "A Model of Memory Contention in a Paging Machine", Comm. ACM, vol. 15 (Aug 1972), pp. 761-771

[Oli74] Oliver, N. A., "Experimental Data on Page Replacement Algorithm", AFIPS Conf. Proc., vol. 43 (May 1974, NCC), pp. 179-184

[OpC74] Opderbeck, H., and W. W. Chu, "Performance of the Page Fault Frequency Replacement Algorithm in a Multiprogramming Environment", Proc. IFIPS Conf., Stockholm, Sweden (Aug 1974), pp. 235-241

[OpC75] Opderbeck, H., and W. W. Chu, "The Renewal Model for Program Behavior", SIAM J. Comput., vol. 4 (Sept 1975), pp. 356-374

[Org72] Organick, E. I., The Multics System: An examination of Its Structure, MIT Press, Cambridge, MA (1972)

[Paz71] Paz, A., Introduction to Probabilistic Automata, Academic, New York (1971)

[Pfi69] Pfister, G., "A MUDDLE Primer?", Doc. SYS.11.01, Programming Technology Division, Project MAC, MIT, Cambridge, MA (July 1969)

[PGL76] Potier, D., E. Gelenbe, and J. Lenfant, "Adaptive Allocation of Central Processing Unit Quanta", J. ACM, vol. 23 (Jan 1976), pp. 97-102

[PPTH72] Parmelee, R. P., T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual Storage and Virtual Machine Concepts", IBM Systems J., vol. 11 (1972), pp. 99-130

[Pra74] Prabhu, N. U., "Stochastic Control of Queueing Systems", Naval Res. Logist. Quart., vol. 21 (Sept 1974), pp. 411-418

[PrF73] Prieve, B. G., and R. S. Fabry, "Evaluation of a Page Partition Replacement Algorithm", Bell Lab. Tech. Report, Naperville, IL (Oct 1973)

[PrF76] Prieve, B. G., and R. S. Fabry, "VMIN - An Optimal Variable-Space Page Replacement Algorithm", Comm. ACM, vol. 19 (May 1976), pp. 295-297

[Ras70] Rasch, P. J., "A Queueing Theory Study of Round-Robin Scheduling of Time-Shared Computer Systems", J. ACM, vol 17 (Jan 1970), pp. 131-145

[R-R72] Rodriguez-Rosell, J., "A Control Approach to Scheduling", Proc. ACM-AICA Int. Computer Symp., Venice, Italy (1972), pp. 561-571

[R-R73] Rodriguez-Rosell, J., "Empirical Working Set Behavior", Comm. ACM, vol. 16 (Sept 1973), pp. 556-560

[R-RD73] Rodriguez-Rosell, J., and J-P. Dupuy, "The Design, Implementation, and Evaluation of a Working Set Dispatcher", Comm. ACM, vol. 16 (April 1973), pp. 247-253

[RyC74] Ryan, T. A., Jr. and E. G. Coffman, Jr., "A Problem in Multiprogrammed Storage Allocation", IEEE Trans. Computer, vol. C-23 (Nov 1974), pp. 1116-1122

[SaK75] Salkin, H. M., and C. A. de Kluyver, "The Knapsack Problem: A Survey", Naval Res. Logist Quart., vol. 22 (March 1975), pp. 127-144

[SaM71] Sage, A. P., and J. L. Melsa, System Identification, Academic, New York (1971)

[Sch73] Schweppe, F. C., Uncertain Dynamic Systems, Prentice-Hall, Englewood Cliffs, NJ (1973)

[Sel65] Selin, I., Detection Theory, Princeton University Press, Princeton, NJ (1965)

[Sha68a] Shapiro, G. F., "Dynamic Programming Algorithms for the Integer Programming Problem - I: The Integer Programming Problem Viewed as a Knapsack Type Problem", Operations Research, vol. 16 (1968), pp. 103-121

[Sha68b] Shapiro, G. F., "Group Theoretic Algorithms for the Integer Programming Problem II: Extensions to a General Algorithm", Operations Research, vol. 16 (1968), pp. 928-947

[ShG69] Shemer, J. E., and S. C. Gupta, "On the Design of Bayesian Storage Allocation Algorithms for Paging and Segmentation", IEEE Trans. Computers, vol. C-18 (July 1969), pp. 644-651

[Shi67] Shinskey, F., Process Control Systems, McGraw-Hill, New York (1967)

[ShS66] Shemer, J. E., and G. A. Shippey, "Statistical Analysis of Paged and Segmented Computer Systems", IEEE Trans. Elect. Computers, vol. EC-15 (Dec 1966), pp. 855-863

[ShT72] Shedler, G. S., and C. Tung, "Locality in Page Reference Strings", SIAM J. Comput., vol. 1 (Sept 1972), pp. 218-241

[SiA61] Simon, H. A., and A. Ando, "Aggregation of Variables in Dynamical Systems", Econometrica, vol. 29 (April 1961), pp. 111-138

[Syn69] Snyder, D. L., The State-Variable Approach to Continuous Estimation, MIT Press, Cambridge, MA (1969)

[SpD72] Spirn, J. R., and P. J. Denning, "Experiments with Program Locality", AFIPS Conf. Proc., vol. 41 (1972 FJCC), pp. 611-621

[Spi76] Spirn, J., "Distance String Models for Program Behavior", Computer, vol. 9, No. 11 (Nov 1976), pp. 14-20

[Spi77] Spirn, J. R., Program Behavior: Models and Measurements, Elsevier, New York (1977)

[The73] Thesen, A., "Scheduling of Computer Programs in a Multiprogramming Environment", BIT, vol. 13 (1973), pp. 208-216

[The75] Thesen, A., "A Recursive Branch and Bound Algorithm for the Multidimensional Knapsack Problem", Naval Res. Logist. Quart., vol. 22 (June 1975), pp. 341-353

[Tru61] Truxal, J. G., "Identification of Process Dynamics", Chap. 3 of Adaptive Control Systems, edited by E. Mishkin and L. Braun, Jr., McGraw-Hill, New York (1961)

[WaW67] Walter, E. S., and V. L. Wallace, "Further Analysis of a Computing Center Environment", Comm. ACM, vol. 10 (May 1967), pp. 266-272

[WeR73] Wells, G. L., and P. M. Robson, Computation for Process Engineers, Wiley, New York (1973)

[Whi63] Whittle, P., Prediction and Regulation, Van Nostrand, Princeton, NJ (1963)

[Wil71] Wilkes, M. V., "Automatic Load Adjustment in Time-Sharing Systems", ACM SIGOPS Workshop on System Performance Evaluation, Harvard Univ., Cambridge, MA (5-7 April 1971), pp. 308-320

[Wil73] Wilkes, M. V., "The Dynamics of Paging", Computer J., vol. 16 (Jan 1973), pp. 4-9

[Wul69] Wulf, W. A., "Performance Monitors for Multi-Programming Systems", Proc. Second ACM Symp. on Operating Systems Principles, Princeton University (1969), pp. 175-181

INITIAL DISTRIBUTION LIST

Addressee	No. of Copies
ASN (RE&S) (W. Smith)	2
ONR, ONR-427, -430	2
CNO, OP-098, -941, -942	3
CNM, MAT-08Y, -08T1	2
NAV SURFACE WEAPONS CENTER, DAHLGREN (R. Hein, J. Miller, W. Warner(2), J. Perry)	5
NAVAIRSYSCOM, AIR-360	1
NAVELECSYSCOM, ELEX 03, PME-108	2
NAVSEASYSCOM, SEA-03C, -032, -034(5), -06H, -63R, -06H2(2), -660, -660F(5), -09G-32 (4), -06V, PME-393	19
NAVOCEANSYSCEN (D. Eddington, R. Kolb, W. Dejka)	3
NAVSEC, SEC-6178D, SEC-6172(3)	4
NAVPGSCOL	1
DDC	12
Prof. Alan Batson, Dept. Comp. Sci., Univ. of Virginia, Charlottesville, VA 22901	1 1
Dr. Forest Baskett, P.O. Box 1663, Los Alamos, NM 87545	1
Mr. Laszlo A. Belady, IBM Watson Res. Center, Yorktowne Heights, NY 10598	1
Prof. Arthur Bernstein, Dept. Computer Science, SUNY Stony Brook, NY 11794	1
Prof. Taylor Booth, Computer Science Dept., Univ. of Connecticut, Storrs, CT 06268	1
Prof. J. C. Browne, Dept. of Comp. Sci., Univ. of Texas, Austin, TX 78712	1
Dr. Jeffrey P. Buzen, BGS Systems Inc., Box 128, Lincoln, MA 01773	1
Prof. Edward G. Coffman, Jr., Dept. of EE & Comp Sci., Univ. of California, Santa Barbara, CA 93106	1
Prof. W. W. Chu, Computer Science Dept., Univ. of California, Los Angeles, CA 90024	1
Mr. William Corwin, Carnegie-Mellon U., Dept. Comp. Sci., Pittsburgh, PA 15213	1
Prof. Peter J. Denning, Computer Science Dept, Purdue Univ., West Lafayette, IN 47907	1
Prof. Jack B. Dennis, MIT, 545 Main St. Cambridge, MA 02139	1
Prof. John J. Donovan, Sloan School of Management, Mass. Institute of Technology, Cambridge, MA 02139	1
Prof. Philip H. Enslow, Jr., School of Information & Computer Science, Georgia Institute of Technology, Atlanta, Georgia, 30332	1
Prof. R. S. Fabry, 1175 Colusa Ave., Berkley, CA 94707	1
Prof. Caxton Foster, Computer and Information Science, U. of Mass. Amherst, MA 01003	1
Prof. W. R. Franta, U. Minn., 143 Space Science Center, 100 Union St., SE, Minneapolis, MN 55455	1
Prof. Ugo O. Gagliardi, Aiken Computation Lab., Harvard University, Cambridge, MA 02138	1
Dr. R. Stocklon Gaines, The Rand Corp., 1700 Main St., Santa Monica, CA 90406	1

INITIAL DISTRIBUTION LIST (CON'T)

Addressee	No. of Copies
Dr. Erol Gelenbe, IRIA-LABORIA, B.P. No. 5, 78150 Le Chesnay, France	1
Dr. Robert P. Goldberg, BGS Systems Inc., Box 128, Lincoln, MA, 01773	1
Dr. Robert L. Gordon, Prime Computer Inc., 145 Penn Ave., Framingham, MA 01701	1
Prof. G. Scott Graham, Univ. of Toronto, CS Research Group, Toronto, Canada	1
Prof. A. Nico Habermann, Carnegie Mellon Univ., 70 Altadena Dr., Pittsburgh, PA 15228	1
Prof. Richard W. Hamming, Computer Science Dept., Naval Post-Graduate School, Monterey, CA 93940	1
Prof. Per Brinch Hansen, Dept. of Computer Science, University of Southern California, Los Angeles, CA 90007	1
Prof. William J. Hemmerle, Dept. of Computer Science and Experimental Statistics, University of Rhode Island, Kingstown, RI 02881	1
Prof. Richard Holt, Computer Sci Dept., Univ. of Toronto, Toronto, Canada	1
Dr. John H. Howard, IBM Research, 5600 Cottle Rd., San Jose, CA, 95123	1
Prof. Edgar T. Irons, Dept. of Computer Science, Yale Univ., 518 Dunham, New Haven, CT 06520	1
Mr. E. Doug. Jensen, Honeywell S&R Center, 2600 Ridgway Parkway, Minneapolis, MN 55413	1
Prof. Anita K. Jones, Dept. of Computer Science, Carnegie-Mellon Univ., Schenley Park, Pittsburgh, PA 15213	1
Ms. Leslie Lamport, Massachusetts Computer Associates, 26 Princess St., Wakefield, MA 01880	1
Dr. Butler W. Lampson, Xerox, 3333 Coyote Hill Rd., Palo Alto, CA, 94306	1
Dr. John Montague, Los Alamos Sci Lab., C-11, MS/296, P.O. Box 1663, Los Alamos, NM 87545	1
Prof. Richard R. Muntz, Computer Science Dept., Univ. of California, Los Angeles, CA 90024	1
Prof. Roger M. Needham, Cambridge University, Computer Cab Corn Exchange St., Cambridge, England	1
Dr. Holger Opderbeck, Telenet Communications Corp., 1666 K Street, N.W., Washington, DC 20006	1
Prof. D. L. Parnas, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill, North Carolina 27514	1
Prof. James L. Peterson, Dept. of Comp Sci, Univ. of Texas, Austin, TX 78712	1
Prof. Gerald Popek, UCLA, 3532 Boelter Hall, Comp Sci Dept, Los Angeles, CA	1
Dr. Dominique Potier, IRIA-LABORIA, B.P. 105, 78150 Le Chesnay, France	1
Dr. Barton G. Prieve, Bell Laboratories, Holmdel, NJ 07733	1

INITIAL DISTRIBUTION LIST (CON'T)

Addressee	No. of Copies
Dr. David P. Reed, M.I.T., NE43-510-545 Technology Square, Cambridge, MA 02139	1
Prof. C. V. Ramamoorthy, Dept. EE&Comp Sci, Univ. of California, Berkeley, CA 94720	1
Dr. Juan Rodriquez-Rosell, IBM, Monterey & Cottle Roads, San Jose, CA 95123	1
Prof. Jerome H. Saltzer, M.I.T., Cambridge, MA 02139	1
Prof. Ashok R. Saxena, University of Colorado, Boulder, CO 80309	1
Prof. Anne Shum, Aiken Computation Lab. Harvard University, Cambridge, MA 02138	1
Prof. Alan Jay Smith, Univ. of Cal-Berkeley, Berkeley, CA 94720	1
Prof. Jeffrey Spirn, Penn State Univ., University Park, PA 16802	1
Prof. Liba Svobodova, M.I.T., 545 Technology Square NE43-513, Cambridge, MA 02139	1
Dr. Robert H. Thomas, Bolt Beranek & Newman, Inc., 50 Moulton St., Cambridge, MA 02138	1
Mr. Chris Tomlinson, Burroughs Corp., P.O. Box 517, Paoli, PA 19301	1
Prof. Dennis Tsichritzis, Stanford Fleming Blg., #10 Kings College Rd., Univ. of Toronto, Toronto, Ont., Canada M5SIA4	1
Prof. Jeffrey D. Ullman, Dept. of Computer Science, Princeton Univ., Princeton, NJ 08540	1
Prof. Andries Van Dan, Center for Computer & Info. Sci., Brown University, Providence, RI 02912	1
Prof. M. V. Wilkes, University of Cambridge, Corn Exchange St., Cambridge CB2 3QG UK	1
Prof. William A. Wulf, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA 15213	1